Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatikdienste
Software Services

ETH Zürich
Benno Luthiger
STC E 13
Stampfenbachstrasse 67
8092 Zürich

Telefon +41 44 632 57 65
benno.luthiger@id.ethz.ch
www.foss.ethz.ch

Software Services (SWS)
Informatikdienste
ETH Zürich

/ch/open-Workshop, 9. September 2014

Workshop-Bericht

Adam Bien „Java EE 7 + Java 8 Good Practices"

Vgl. http://www.adam-bien.com/roller/abien/

Relevante Hinweise aus Adam Biens Präsentation:

Externe Abhängigkeiten sollten auf Minimum reduziert werden
Hat eine Anwendung eine externe Abhängigkeit (z.B. ein JAR-File mit einer gewissen Funktionalität), so sollten vor jedem Release der Anwendung folgende Schritte durchgeführt werden:

1. Prüfen, ob eine neue Version des externen Moduls existiert.
2. Falls neue Version vorhanden: prüfen, ob die neue Version ein Sicherheits- oder Performance-Problem löst.
3. Falls ja: neue Version einbauen.
4. Falls neue Version eingebaut: Integrations- und Systemtests durchführen, damit sichergestellt ist, dass die neue Version nicht zu unerwarteten Effekten führt.

Dieser Prozess, welcher wichtig ist für die langfristige Stabilität der Anwendung, ist recht aufwendig. Längerfristig kann ein solcher Aufwand den anfänglichen Nutzen (Produktivitätsvorteil) überwiegen. A. Bien reduziert aus diesem Grund den Einsatz von externen Modulen auf ein Minimum. D.h. er setzt solche Module nur dort ein, wo sich notwendig sind (z.B. Erzeugen von PDF-Dokumenten aus einem Text).

Sicheres Deployment von Enterprise-Anwendungen
Das Ausbreiten einer Anwendung wird Narrensicher, wenn nicht nur die Anwendung (als WAR-File), sondern der ganzen Stack ausgeliefert wird, d.h. das WAR-File im App-Server im (virtuellen) Betriebssystem. Docker (www.docker.com) ist ein virtuelles Betriebssystem, welches diesen

Vorgehensfall stark vereinfacht. Mit diesem Hilfsmittel können virtualisierte Linux-Container mit abgeschlossenen und eigenständigen Anwendungen erzeugt werden. Weil alle Abhängigkeiten im Docker-File enthalten sind, funktioniert ein Docker-File, welches auf einer Linux-Installation läuft, auch auf jeder anderen Linux-Installation. Ein Deployment mit Hilfe von Docker verläuft deshalb sehr einfach: Das Docker-File auf den Linux-Server installieren und starten. Falls die Anwendung nicht läuft, das Docker-File stoppen und die letzte laufende Version des Docker-Files wieder starten. Docker ist ein wichtiges Hilfsmittel, damit Continuous Delivery realisiert werden kann.

REST

REST ist eine sehr schnelle Schnittstelle, d.h. eine Anwendung kann über eine REST-Schnittstelle die Informationen sehr rasch bekommen.
Bei REST sind alle Aufrufe ausser POST indempotent, d.h. ohne Nebenwirkungen. Ein idempotenter Aufruf kann beliebig oft hintereinander aufgerufen werden und führt zum selben Ergebnis wie der erste Aufruf.
REST spielt eine grosse Rolle für die Client-Server-Kommunikation bei mobilen Anwendungen. Eine REST-Schnittstelle kann für mobile Anwendungen sehr leicht entwickelt werden. Eine SOAP-Schnittstelle ist in diesem Gebiet viel komplizierter. SOAP funktioniert einigermassen gut nur als Kommunikation von Java zu Java.
Wird REST für eine externe Schnittstelle eingesetzt, so sollen die Informationen in Form von JSON oder XML ausgegeben werden. Wird REST als Schnittstelle zwischen Java-Applikationen oder –Modulen verwendet, so sollen die Java-Objekte serialisiert werden (z.B. mit Kryo, https://github.com/EsotericSoftware/kryo).
Typsicherheit mit REST: Bei einer externen Schnittstelle werden die Daten als JSON-Objekte formatiert. Typsicherheit ist damit nicht gegeben, was aber kein grosses Problem ist. Wichtig ist, dass beim Weiterentwickeln der Schnittstelle keine Attribute wegfallen, sondern immer nur neue Attribute dazu kommen (damit die alten Clients der REST-Schnittstelle weiter funktionieren).
Die REST-Schnittstelle einer Applikation soll mit geeigneten Werkzeugen selbstdokumentierend und maschinenlesbar gemacht werden.
Z.B. stripe (https://stripe.com/) oder Swagger (http://swagger.io/)

DAO sind unnütz

Gemäss Bien sind Data Access Objects (DAOs) seit der Einführung von JPA in Java nicht mehr notwendig. DAOs kapseln den Zugriff den Datenzugriff (JDBC) hinter einer Implementierung, welche ersetzt und gemockt werden kann. Mit JPA wird jedoch ein EntityManager injiziert, welcher genau dies auch leistet.

BCE-Pattern und Paketstruktur

In einem JEE-Projek soll die Fachlichkeit in der Projektstruktur abgebildet werden. Ein Java-Paket *.exc* für Exceptions bildet keine Fachlichkeit ab, sondern Programm-Artifakte. Das ist allenfalls für ein Framework sinnvoll.
Jedes fachliche Paket besteht in einer sinnvollen Struktur aus drei Unterpaketen. Diese entsprechen dem Boundary-Control-Entity-Pattern (BCE). Das BCE-Muster entspricht grob dem MVC-Muster.
In das *Boundary*-Paket kommt alles, was mit der Schnittstelle zu tun hat, insbesondere die EJB für die Transaktionen und den EntityManager sowie der Code für eine allfällige REST-Schnittstelle. Die EJB ist zustandslos (@Stateless).

Wird die EJB so gross, dass sie nicht mehr gut gewartet werden kann, so werden gewisse Aufgaben in das *Control*-Paket ausgelagert. In diesem Paket befinden sich ausschliesslich POJOs, welche in die Bean injiziert werden. Diese Klassen sind immer zustandslos.
Das *Entity*-Paket ist für die Fachlichkeit und Persistenz zuständig.
Generell gilt: alles was ein Service ist, soll injiziert werden (new ist verdächtig), alles was Objekt-orientiert ist, soll instanziiert werden.

Projekt != Produkt
Ein Produkt muss generischer sein als ein Projekt. Ein Projekt hat einen spezifischen Kunden und muss genau dessen Anforderungen erfüllen. Bei einem Produkt kommen die Kunden meist erst, wenn die Arbeiten (an der ersten Version des Produkts) abgeschlossen sind.

Lasttest != Stresstest
Ein Lasttest simuliert üblicherweise das Verhalten des Systems unter realistischen Einsatzbedingungen. Entsprechend enthält ein Lasttest „Denkpausen“ zwischen den Aufrufen. Ziel eines Lasttest ist es, herauszufinden, wieviel Last die Anwendung verkraften kann, bis sie einbricht.
Mit einem Stresstest will man die Anwendung in jedem Fall zur Überlastung bringen. Ziel des Stresstests ist es, herauszufinden, welches die Schwachstellen des Systems sind, wenn es unter grosse Last gesetzt wird. Mit einem Stresstest soll insbesondere getestet werden, ob die bei der Implementierung getroffenen Annahmen bzgl. Mitläufigkeit und Race-Conditions korrekt sind.

Luthiger Benno (ID SWS)