Belegsystem 2.0

Schlussbewertung

Technologien

Java

Wir haben uns für die objektorientierte Sprache Java entschieden, weil die Projektmitglieder bereits gute Erfahrungen mit dieser Sprache sammeln konnten. Zu Beginn des Projekts hatten wir noch überlegt, das Belegsystem in Python bei Verwendung des Django-Frameworks zu programmieren. Wir haben uns dann aber aufgrund der schon vorhandenen Expertise eines Teammitgliedes für die Verwendung von Apache Tapestry als grundlegendes Web- und Application-Framework für unser Projekt entschieden. Tapestry selber ist in Java geschrieben, daher lag es nahe auch in Java zu programmieren (es ist aber auch möglich z.B. Scala mit Tapestry zu verwenden).
Die Arbeit mit Java selbst verlief flüssig und stellte uns vor keine größeren Probleme. Auch wenn Java nicht die Möglichkeiten einer modernen objektorientiert-funktionalen Sprache wie Scala bietet so waren wir dennoch sehr angetan, was es in Java für Möglichkeiten der erweiterten bzw. Meta-Programmierung duch Generics und Reflections gibt, was wir für die grundlegende Struktur unserer Anwendung intensiv eingesetzt haben.

Tapestry

Wir haben uns wie oben bereits erwähnt für das Open-Source-Web-Framework Apache Tapestry entschieden (http://tapestry.apache.org/).
Für Tapestry sprach letztendlich, dass wir alle Java-Erfahrungen einbringen konnten, ein Teammitglied sogar bereits über sehr gute Tapestry-Kenntnisse verfügte und Tapestry, verglichen mit anderen Frameworks, beste Werte in Punkto Performance und Skalierbarkeit lieferte (siehe http://www.jtict.com/blog/rails-wicket-grails-play-lift-jsp/)
Bei der Arbeit von Tapestry können wir von zwei unterschiedlichen Erfahrungen berichten: Die Neueinsteiger in die Arbeit mit dem Framework waren begeistert davon, wie einfach sich Webprogrammierung damit erledigen lässt. Zu dem guten Einstieg hat neben der guten Dokumentation und einer Seite mit sehr viele Tapestry-Beispielen inklusive ausführlicher Erläuterung (http://jumpstart.doublenegative.com.au/jumpstart/) auch die Hilfe durch das bereits mit Tapestry vertraute Teammitglied beigetragen. Dies hat die Arbeit mit dem Framework doch sehr erleichtert und es möglich gemacht, dass wir alle angedachten Features auch so umsetzen konnten.
Ein Teammitglied (Dominik) hatte wie erwähnt bereits umfangreiche Kenntnisse in Tapestry. Für ihn lag der Reiz darin, im Rahmen dieses Projektes anderen Menschen das Framework nahe zu bringen und für eine gut funktionierende grundlegende Struktur und Basis der Anwendung zu sorgen. Dabei war er ebenfalls begeistert - und zwar von den Möglichkeiten die Tapestry hinsichtlich Meta- bzw. aspektorientierter Programmierung (z.B. mithilfe der Verwendung von eigenen Annotations) und zusätzlicher Erweiterbarkeit bietet (z.B. wie leicht es war einen AccessFilter zu schrieben der selbst Methodenzugriff überwacht, oder auch wie unkompliziert es war Tapestry um unser eigenes Exception Reporting zu erweitern).

JPA/Hibernate

Wir haben unsere Klassen mit JPA-konformen (Java Persistence API) Annotationen erweitert so dass sie von einem JPA-konformen Layer persistiert werden können. Als konkreten Persistente-Layer verwenden wir schließlich Hibernate, da es zum einen das

Modul tapestry-hibernate gibt, mithilfe dem sich Tapestry um Session-Verwaltung und grundlegende Einbindung von Hibernate usw. kümmert und wir das nicht erledigen müssen. Außerdem bringt Hibernate die Hibernate Query Language (HQL) mit mithilfe derer sich Abfragen leichter als mit SQL schreiben lassen (da näher an der Objektwelt). Dies haben wir für manche spezielle Abfragen verwendet, manche Abfragen wurden aber auch mittels der Criteria API erstellt (ursprünglich ebenfalls Hibernate-spezifisch, seit JPA 2.0 auch Bestandteil des Standards).
Mithilfe von JPA/Hibernate war es kein Problem die Persistenz unserer Anwendung zu gewährleisten. Im Gegensatz zu der Erfahrung in manch anderen Teams hatten wir im Zusammenspiel von Hibernate und Tapestry auch keine Probleme mit den Sessions bzw. mit schon in der Session vorhandenen Objekten. Auch das Nachladen von Collections (da lazy-Fetch-Mode) stellte kein Problem dar.

MySQL

Aufgrund der weiten Verbreitung haben wir uns für eine MySQL-Datenbank entschieden. Es hätte aber auch jede andere sein können, da Hibernate den Zugriff auf jede Datenbank durch einfaches ändern der Konfiguration-XML-Datei ermöglicht. MySQL war insofern für uns nicht direkt wahrnehmbar, da hier keine weiteren Einstellungen vorgenommen werden mussten.

JUnit

Um eine umfangreiche Testsuite zu erstellen, haben wir auf das Java-Test-Framework JUnit zurückgegriffen. Dabei haben wir in den einzelnen Schichten jeweils für die einzelnen Klassen eigene Testklassen geschrieben und hier für jede Methode einen eigenen Testfall beschrieben. Um wirklich alle Schichten zu testen haben wir auch Oberflächentests geschrieben.
Die Tests können sowohl in der Entwicklungsumgebung als auch über Maven von der Konsole aus gestartet werden.
Wir müssen im Nachhinein zugeben dass wir zwar am Ende relativ viele Tests geschrieben haben, aber zwischendurch die Überprüfung mittels Testklassen etwas vernachlässigt haben. Dies war für unsere Anwendung kein Problem, generell empfiehlt es sich aber, mehr Augenmerk auf ein frühzeitiges Testen der Anwendung zu richten.

MulTex

Als Exception Handling Framework haben wir MulTex mit in unser Projekt integriert.
Es bietet viele Möglichkeiten zur zentralen Ausnahmebehandlung wie zum Beispiel der Internationalisierung bei Fehlermeldungen, übersichtlicherer Fehlerzurückverfolgung und einer standardisierten Fehlerbehandlung in den Methoden.
Wir haben MulTex aber noch erweitert - und zwar um die Klasse *AnnotatedExc* und *AnnotatedFailure* (die *Exc* und *Failure* erweitern), bei denen mittels der Annotation *@ExcMessage* die Exception-Message gesetzt werden kann und nicht nur als Javadoc-Kommentar (daran ist u.a. praktischer, dass der Text dann gleich von IDE&Compiler als String-Literal erkannt wird was dann auch bei mehrzeiligen Texten gut funktioniert).
Dadurch waren wir in der Lage, selbst *ohne* dem bei MulTex vorgesehenen Einsammeln der Ausnahmetext über ein spezielles Ant-Skript gleich die passenden Meldungstexte beim Auftreten einer annotierten Exception (die also von *AnnotatedExc* erbt) darzustellen.
Um aber die Funktionalität des Einsammeln der Texte (um sie dann z.B. lokalisieren zu

lassen) dennoch zu gewährleisten haben wir eine eigene Klasse geschrieben die sich darum kümmert - die *ExcMessagesToProperties.* Diese sammelt per Reflection alle Klassen ein die mit der ExcMessage-Annotation versehen ist und erstellt daraus eine Properties-Datei die dann zur Übersetzung gegeben werden kann.
Zusätzlich unterstützen wir, dass die properties-Dateien nicht im Classpath liegen müssen sondern z.B. im WEB-INF-Ordner. Beim *Startup* des Systems werden sie zur Laufzeit als ResourceBundle eingebunden.
Durch diese Erweiterungen fiel die Verwendung von MulTex zum Exception Handling sehr leicht.

Werkzeuge

Eclipse

Zur Entwicklung unserer Projektes haben wir die quelloffene Entwicklungumgebung **Eclipse** verwendet, weil sie einfach zu erweitern ist (z.B. Plug-ins für *Maven, Subversion, Jetty*) und alle Teammitglieder im Umgang mit *Eclipse* bereits geübt waren. In **Eclipse** konnten dann nicht nur sämtliche Quelldateien erstellt und editiert werden, das Projekt konnte von hier aus auch immer gleich ausgeführt und umfangreich getestet werden. Für jeden Datentyp stellt Eclipse bzw. ein passendes Plugin dann eine gute Ansicht zur Verfügung.
Dank der bei modernen *IDE*s üblichen Funktionen wie Autovervollständigung, automatischer Formatierung, etc. konnten die Dateien bequem, schnell und übersichtlich erstellt werden. Eclipse ist ***die*** IDE für Java-Developer, da gibt es nichts dran zu rütteln.

Subversion (SVN)

Zu Beginn des Projekts haben wir ein ***SVN-Repository*** bei *assembla.com* registriert. Über ein Plug-in in Eclipse konnten wir dann unsere Änderungen und Erweiterungen den anderen Teammitgliedern direkt zur Verfügung stellen. Sollte das Programm nach einem Update dann nicht mehr funktionieren, konnte man die Daten ohne weiteres auf eine frühere Version zurücksetzen bzw. zur Fehlersuche den Quellcode direkt mit einander vergleichen.
Dateien wie UML-Diagramme haben wir ebenfalls über das Repository ausgetauscht.
Die Verwendung eines Tools zur Versionkontrolle und -management hat sich als sehr sinnvoll herausgestellt, ohne wäre das Projekt bzw. das verteilte Arbeiten auch gar nicht möglich gewesen!

Maven

Wir haben **Apache Maven** als Build-Management-Tool eingesetzt - und zwar von Anfang an. Von Tapestry gibt es nämlich einen Maven-Archetype auf dessen Grundlage wir angefangen haben unsere Anwendung zu entwickeln. Wir haben Maven in erster Linie zur Auflösung von Abhängigkeiten benutzt (über Einträge in der pom.xml) - wofür es sehr hilfreich war.
Etwas Schwierigkeiten hatten wir mit der Einbindung des passenden Maven-Eclipse Plugins (m2eclipse), da wir erst noch eine auf Maven 2.0 basierende Version benutzt hatten, die neuere Version (unter Obhut der Eclipse-Foundation) aber schon Maven 3.0 voraussetzte.

Visual Paradigm

Zum Erstellen unserer UML-Diagramme haben wir das proprietäre Programm *Visual*

Paradigm 8.0 (VP8) verwendet. Lizenzen für die Standard Version konnte uns vom Fachbereich VI der Beuth-Hochschule zur Verfügung gestellt werden. *VP8* stellt mehr Funktionen als vergleichbare Open-Source Programme zur Verfügung und ist leichter zu bedienen, weshalb wir hier ein einziges Mal auf nicht freie Software zurückgegriffen haben.

Fazit

Die verwendeten Technologien und Werkzeuge waren für uns alle gut handhabbar, wobei es natürlich von großem Vorteil war, schon mit den Technologien vertraute Personen in der Gruppe zu haben. Aber gerade dadurch gelang uns ein guter und tiefer Einblick in die verwendeten Technologien.
V.a. die Wahl von Tapestry hat sich als besonders gut erwiesen - wir waren immer wieder angetan von den Möglichkeiten des Frameworks und wie einfach sich damit viele Sachen realisieren lassen.
Die Zusammenarbeit im Team lief auch gut. Wir haben meistens über Mail & Skype kommuniziert und ansonsten verteilt gearbeitet. Zweimal haben wir uns auch jeweils einen ganzen Tag getroffen und zusammen am Belegsystem gearbeitet.
Team-Besprechungen haben wir immer gleich im Anschluss an die Rücksprachen abgehalten. Wir haben kein zusätzliches Projekt-Management-Tool verwendet, haben es aber zusammen dennoch geschafft den Überblick über die zu erledigenden Aufgaben zu halten. Bei größeren Projekten empfiehlt sich aber deren Einsatz, unseres war für das Festhalten von Aufgaben in Mails und Task-Listen noch gerade so ausreichend.

Über einen Aspekt sind wir in der Rückschau aber etwas enttäuscht (wir sind uns aber mittlerweile über auch die Ausmaße eines solchen Vorhabens mehr im Klaren):
Ursprünglich war unsere Motivation für dieses Projekt darin gelegen, tatsächlich für doiese Hochschule ein anderes Belegsystem zu entwickeln da das bestehende System langsam, unausgegoren und teilweise kaum benutzbar für uns Studierende ist.
Zunächst begann das Projekt für uns ziemlich vielversprechend, wir haben einen guten Entwurf hierfür vorgelegt und es gab im Rahmen von Softwareprojekt 1 auch weitreichende Überlegungen wie wir dieses Projekt auch der Hochschulleitung gegenüber schmackhaft machen können.
Enttäuscht mussten wir feststellen, dass es aber keine Rolle spielt, wie gut unser Projekt wird, weil aufgrund der Hochschulpolitik keine Chance besteht (so jedenfalls die Aussage von Professoren), dass unser Projekt hier jemals zum Einsatz kommt.
Aber davon haben wir uns nicht entmutigen lassen und in unseren Augen ein sehr gutes Endresultat umgesetzt: Es ist performant, einfach zu bedienen, sicher und schlank. Optisch ist es zwar momentan auf die Beuth-Hochschule zugeschnitten, es kann jedoch auch ohne weiteres in anderen Einrichtungen eingesetzt werden.
Daher können wir viel Positives aus diesem Projekt mitnehmen. Auch war es mal eine Erfahrung, eine Nacht durch zu arbeiten (um für die Rücksprache alles fertig zu bekommen).
Zusammenfassend können wir also sagen, dass wir viel gelernt haben und auch etwas stolz auf unser Projekt sind, in das wir sehr viel Arbeit investiert haben.