



EvaJ-Projektdokumentation

Dieses Dokument enthält die EvaJ-Projektdokumentation. Es gibt Auskunft über verschieden Aspekte des am Hasso-Plattner-Institut für IT-Systemtechnik entwickelten und installierten Evaluierungssystems EvaJ.

Inhalt der Projektdokumentation sind verschiedene Aspekte der Architektur von EvaJ, Betrachtungen zur Projektdurchführung, Erfahrungsberichte zur Entwicklung und schließlich Erfahrungsberichte über EvaJ im Einsatz.

Dieses Dokument wurde von Martin Breest, Gero Decker, Volker Gersabeck, Silvan Golega, Lukas Neitsch, Jan Schaffner und Jan Schulz-Hofen für das während der Übung „Trends und Konzepte in der Softwareindustrie“ entwickelte Evaluierungssystem EvaJ erstellt. Die Übung wurde im Wintersemester 2005/06 am Hasso-Plattner-Institut für IT-Systemtechnik an der Universität Potsdam durchgeführt.

Inhaltsverzeichnis

1 Architektur	5
1.1 Die Architektur von EvaJ	5
1.2 Das Datenmodell von EvaJ	8
Verwaltung der Evaluierungen	8
Verwaltung von Fragebögen	10
Verwaltung der Veranstaltungsbelegungen der Studenten	11
Verwaltung der einzelnen Bewertungen	11
Verwaltung der aggregierten Veranstaltungsergebnisse	12
Verwaltung allgemeiner Aspekte	12
1.3 Sicherheitsaspekte	13
Zugriffsschutz via Kerberos	13
Kommunikation zwischen Frontend und Backend	15
2 Projektdurchführung	16
2.1 Projektstruktur	16
2.2 Vorgehensmodell	16
Analyse	17
Design	17
Implementierung	17
Frontend	17
Backend	18
Test	18
Einsatz	18
2.3 Herausforderungen Fernarbeit	19
3 Erfahrungen zur Entwicklung	20
3.1 Erfahrungen mit den verwendeten Technologien	20
JSF	20
Hibernate und Spring (TODO)	21
3.2 Erfahrungen mit der Entwicklungsumgebung	21
4 Erfahrungen zu EvaJ im Einsatz	23
4.1 Technische Herausforderungen	23

4.2 Organisatorische Herausforderungen	23
4.3 Funktionale Herausforderungen	24
4.4 Nutzungsprofile	24

1 Architektur

Das EvaJ Projekt ist eine Umsetzung des Eva-Designs aus dem ersten Teil der Übung „Trends und Konzepte“ am Lehrstuhl für Enterprise Platform and Integration Concepts (EPIC) des Hasso-Plattner-Instituts. Diese Umsetzung wurde mithilfe von Java Technologie durchgeführt, weshalb auch der Projektname EvaJ gewählt wurde.

Dieses Dokument bietet einen Überblick über das entwickelte System, beschreibt die Durchführung des Projekts mit ihren speziellen Herausforderungen sowie erläutert einige Erfahrungen bezüglich der Entwicklung und des Einsatzes von EvaJ. Des Weiteren sind zusammen mit dieser Dokumentation eine Anleitung zur Administration des Systems sowie zur Benutzung erhältlich.

1.1 Die Architektur von EvaJ

Aus dem ersten Teil der Übung ging ein Design-Konzept hervor, welches es umzusetzen galt. Vorgeschrieben waren nur die Oberflächen, die die einzelnen Nutzer des Systems zu sehen bekommen sollen und die Navigation zwischen diesen Oberflächen. Für die genaue Umsetzung gab es vorerst keine weiteren Vorgaben aus dem ersten Teil. Für dieses Projekt gab es dann die Aufgabe, Java-Technologie für die Umsetzung zu wählen.

Da es sich hierbei um eine Web-Anwendung handelt, wurde eine 3-Schichten-Architektur gewählt, wie sie in Abbildung 1 zu sehen ist. Die in farblich unterlegten Teile bilden dabei die 3-Schichten-Architektur. Die übrigen dargestellten Akteure werden im folgenden erläutert:

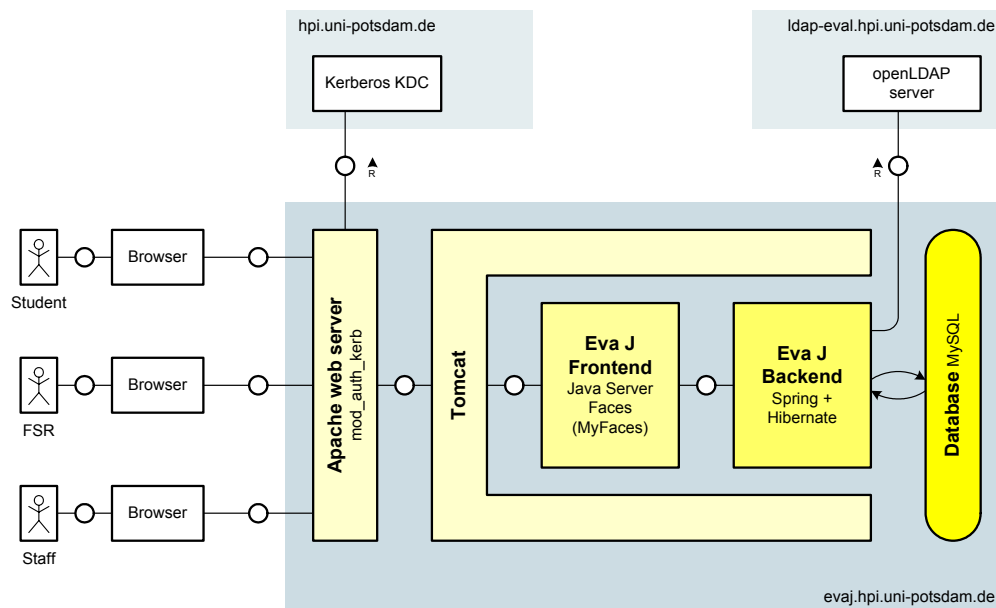


Abbildung 1: Systemlandschaft von EvaJ.

Jede Nutzergruppe greift auf das System mithilfe eines einfachen Browsers zu, so dass die Client-Seite keines Entwicklungsaufwands bedarf. Auf dem EvaJ-Server wurden zwei Web-Server installiert, die das System verwalten. Zum einen ist dies ein Apache-Web-Server, der um das Modul `mod_auth_kerb` erweitert wurde. Diese Konfiguration dient der Authentifizierung der Nutzer gegenüber einem Kerberos-Server. Auf die Sicherheitsaspekte wird

später noch näher eingegangen. Nach erfolgter Authentifizierung werden alle Anfragen an einen Tomcat-Web-Server weitergeleitet, der das eigentliche System ausführt. Innerhalb dieses Servers befinden sich der Frontend- und der Backend-Teil von EvaJ

Für die Datenhaltung wurde eine MySQL Datenbank installiert, auf die das Backend Zugriff hat. Zudem hat das Backend die Möglichkeit bei einem openLDAP-Server die Rollen der einzelnen Nutzer zu erfragen. Somit wird sichergestellt, dass die Nutzer nur die ihrer Rolle entsprechende Funktionalität aufrufen können. Im folgenden Abschnitt wird nun detaillierter auf die einzelnen Komponenten des Systems und ihre Interaktionen eingegangen.

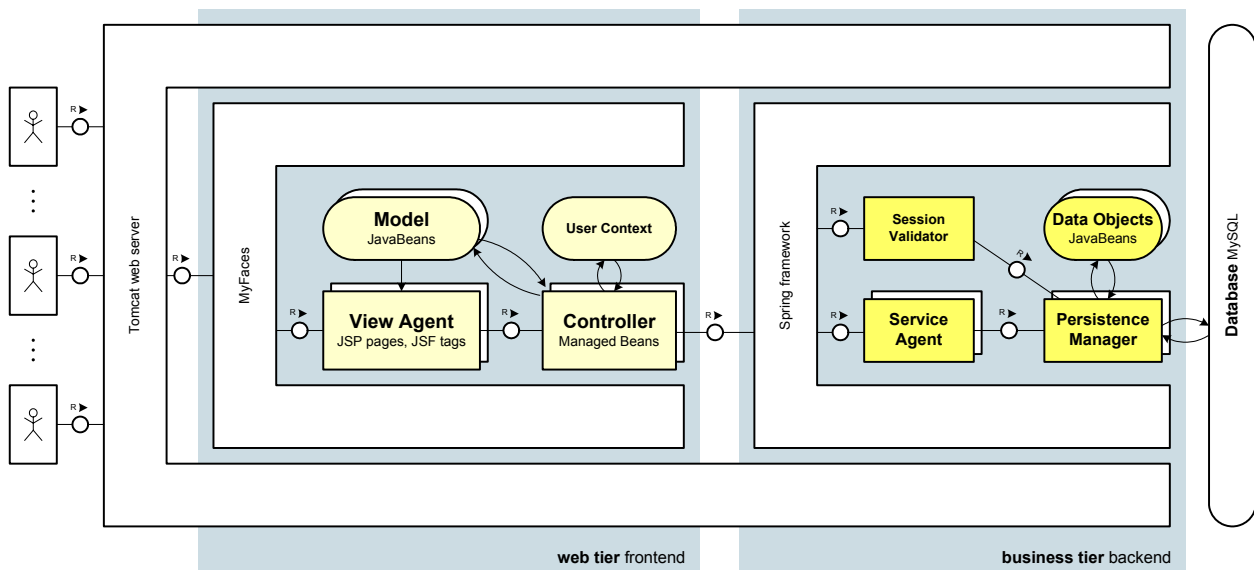


Abbildung 2: Architektur von EvaJ.

Abbildung 2 zeigt die detaillierte Architektur von EvaJ, speziell den Aufbau von Frontend und Backend. Bei der Umsetzung des Frontends wurden JavaServerFaces in der MyFaces-Implementierung eingesetzt. Im Einzelnen bedeutet dies, dass JavaServerPages mit speziellen JSF-Tags verwendet wurden, um die verschiedenen, vom Eva-Design vorgegebenen Oberflächen zu erstellen. Diese Views nutzen einfache JavaBeans um dynamisch die verschiedenen Daten einzubinden. Sogenannte „Managed Beans“ sind dafür verantwortlich die JavaBeans zu erstellen und mit den richtigen Daten zu bestücken. Die Managed Beans bilden die Controller, die aus dem Model-View-Controller-Prinzip bekannt sind. In diesem Sinne wird das Model der MVC-Architektur durch die JavaBeans repräsentiert.

Für die Navigation zwischen den einzelnen Seiten gibt es in dem benutzten Framework folgenden Mechanismus: Wird auf einer Seite eine Aktion ausgeführt, so wird eine Methode im dazugehörigen Controller Objekt aufgerufen. Diese Methode ist ausschließlich für die Behandlung der Aktion zuständig, sie speichert also z.B. eingegebene Daten im Backend. Als Ergebnis liefert die Methode dann eine Zeichenkette, anhand derer die Seite ermittelt wird, die als nächstes angezeigt werden soll. Somit kann die gesamte Navigation mit einem Satz Regeln für diese Zeichenketten umgesetzt werden, ohne dass die Controller-Methoden im Einzelnen die Folgeseiten kennen müssen.

Über ein gemeinsam genutztes User-Context-Objekt stellen die Controller sicher, dass vom Backend immer die diesen Nutzer betreffenden Daten abgeholt werden. Bei jeder Backend-Anfrage wird dieses Objekt mit übermittelt.

Für die Kommunikation zwischen Front- und Backend sowie die Verwaltung von Letztere wurde das Spring-Framework eingesetzt. Der Vorteil dieses Frameworks ist die Abstraktion für den Entwickler von der letztendlich eingesetzten Kommunikationstechnologie. Mithilfe einer kleinen Änderung in der Konfiguration ist es möglich, z.B. an Stelle von Web-Service-Aufrufen normale Java-Aufrufe zu benutzen. Somit kann man leicht zwischen umfangreicher Interoperabilität durch Web services auf der einen Seite und wesentlich performanteren Java-Aufrufen auf der anderen Seite wählen, je nach gewünschten Anforderungen.

Ein weiterer Vorteil des Spring-Frameworks ist der Einsatz von so genannten Interceptoren. Diese ermöglichen ohne Änderung der eigentlichen Services ein Vorschalten von verschiedenen Überprüfungen. So wird bei EvaJ vor jedem Aufruf überprüft ob das im User Context mitgelieferte Session Objekt noch gültig ist. Sollte dies nicht der Fall sein, wird der Aufruf einfach an dieser Stelle unterbunden. In der Implementierung der einzelnen Services ist von dieser Überprüfung nichts zu sehen, weshalb der Entwickler sich auf die Umsetzung der Services konzentrieren kann.

Bei der Datenhaltung wurde mit Hibernate eine Technologie eingesetzt, die es erlaubt von der eingesetzten Datenbank im Hintergrund zu abstrahieren. Mithilfe eines Objekt-Relationalen Mappings ist es möglich im Backend mit simplen JavaBeans zu arbeiten. Wie diese Objekte jedoch in der relationalen Datenbank abgelegt werden ist für den Entwickler transparent. Dies ermöglicht auch den Einsatz verschiedener Datenbanken, wie z.B. eine In-Memory-Datenbank während der Entwicklung und eine MySQL-Datenbank für den realen Einsatz von EvaJ

Bevor das Datenmodell im Detail vorgestellt wird, ist noch eine zentrale Sache erwähnenswert: die Schnittstelle zwischen Front- und Backend. Die Definition dieser Schnittstelle war einer der ersten Schritte in der Entwicklung von EvaJ Dank gutem Design konnte die Schnittstelle mit nur minimalen Anpassungen während der gesamten Entwicklung genutzt werden. An dieser Stelle soll jedoch auf eine genaue Beschreibung verzichtet werden; diese kann man in den JavaDocs im Einzelnen nachlesen.

1.2 Das Datenmodell von EvaJ

Das EvaJ zugrunde liegende Datenmodell ist im ER-Modell in Abbildung 2 dargestellt.

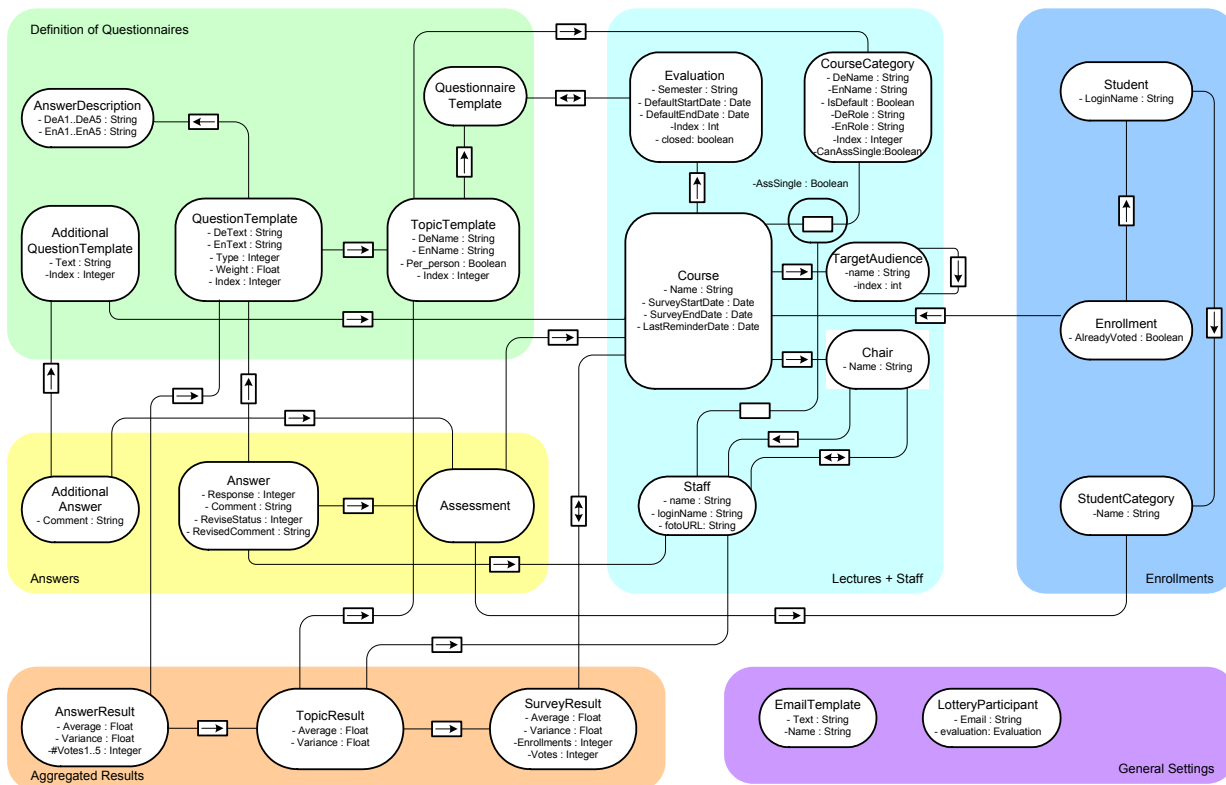


Abbildung 3: ER-Modell des EvaJ zugrunde liegenden Datenmodells.

Wie man sehen kann, sind die dargestellten Entitäten sechs Teilbereichen zugeordnet. Diese Teilbereiche sind:

1. die Verwaltung der Evaluierungen, der dazugehörigen Veranstaltungen und des Personals (Bereich *Lectures + Staff*),
2. die Verwaltung der Fragebögen (Bereich *Definition of Questionnaires*),
3. die Verwaltung der Veranstaltungsbelegungen der Studenten (Bereich *Enrollments*),
4. die Verwaltung der einzelnen Bewertungen (Bereich *Answers*),
5. die Verwaltung der aggregierten Evaluierungsergebnisse für veröffentlichte Evaluierungen (Bereich *Aggregated Results*) und letztlich
6. die Verwaltung allgemeiner Aspekte (Bereich *General Settings*).

Im Folgenden werden diese Bereiche genauer erläutert.

Verwaltung der Evaluierungen

EvaJ unterstützt die Verwaltung einer beliebigen Anzahl von Evaluierungen. Evaluierungen werden im Datenmodell durch die Entität *Evaluation* repräsentiert. Jede Evaluierung läuft für ein bestimmtes Semester und hat deshalb ein festes Start- und Enddatum. Zudem wird zwischen noch laufenden und bereits abgeschlossenen Evaluierungen unterschieden. Die Reihenfolge in der die Evaluierungen durchgeführt wurden, ist durch einen den Wert des At-

tributes *index* ersichtlich.

Jeder Evaluierung kann eine beliebige Anzahl von zu bewertenden Lehrveranstaltungen hinzugefügt werden. Lehrveranstaltungen werden durch die Entität *Course* im Datenmodell repräsentiert. Jede Lehrveranstaltung hat einen Namen (z.B. Business Process Management II). Zudem kann für jede Lehrveranstaltung ein Start- und Enddatum für die Evaluierung angegeben werden. Diese Daten überschreiben die in der Evaluierung vorgegebenen Standardwerte. Da für Lehrveranstaltungsevaluierungen auch beliebig viele Erinnerungen verschickt werden können und man den Überblick über die verschickten Erinnerungen behalten möchte, kann das Datum der letzten Erinnerung im Attribut *LastReminderDate* gespeichert werden.

Der Typ einer Veranstaltung kann grundsätzlich frei konfiguriert werden. Es werden Vorlesungen, Seminare, Übungen und Projekte standardmäßig unterstützt. Es können aber beliebig viele Veranstaltungstypen festgelegt werden. Ein Veranstaltungstyp wird im Datenmodell durch die Entität *CourseCategory* repräsentiert. Jeder Veranstaltungstyp hat einen deutschen und einen englischen Bezeichner, wie zum Beispiel „Vorlesung“ und „Lecture“ und eine deutsche und englische Bezeichnung für die Rolle der beteiligten Personen, wie zum Beispiel „Dozent“ und „Lecturer“. Um weitere Sprachen unterstützen zu können, muss das Datenmodell entsprechend erweitert werden. Man kann zudem festlegen, welcher Veranstaltungstyp Standard sein soll. Über das Attribute *CanAssSingle* kann auch festgelegt werden, ob bei der Veranstaltungsbewertung alle oder nur ein Person bewertet werden muss.

Durch die Verknüpfung von Lehrveranstaltungen mit bestimmten Veranstaltungstypen kann der Typ der Veranstaltung festgelegt werden. Dabei können beliebig viele Verknüpfungen erstellt werden, um nicht nur einfache Veranstaltungstypen wie „Seminar“ oder „Vorlesung“ zu ermöglichen, sondern auch Kombinationen wie „Vorlesung mit Übung“ oder „Vorlesung mit Seminar“. Die Verknüpfungen von Lehrveranstaltungen mit Veranstaltungstypen wird im Datenmodell durch die Entität *CourseCategoryMapping* repräsentiert.

Die Dozenten, Seminarleiter und Übungsleiter und weiteres Personal, das an bestimmten Veranstaltungen beteiligt ist, kann durch die Verknüpfung beliebiger Mitarbeiter mit einem *CourseCategoryMapping* festgelegt werden. Eine Verknüpfung von „Prof. Weske“ mit einem *CourseCategoryMapping* zwischen der Veranstaltung „BPM II“ und dem Veranstaltungstyp „Vorlesung“ legt demnach fest das Prof. Weske die Rolle Dozent für diese Veranstaltung übernimmt. Würde er als Mitarbeiter in einem zu bewertenden Projekt eingetragen werden, würde er z.B. die Rolle Projektleiter übernehmen.

Jede Veranstaltung hat unterschiedliche Zielgruppen, die diese Veranstaltung belegen können. Grundsätzlich kann man am HPI zwischen Bachelor- und Masterstudenten unterscheiden. Es sind aber auch andere Hörerschaften denkbar. Zielgruppen werden im Datenmodell durch die Entität *TargetAudience* repräsentiert. Es können beliebig viele Zielgruppen festgelegt werden. Die Namen für diese Zielgruppen wären dann z.B. „Bachelor“ oder „Master“. Die potentielle Hörerschaft, die eine Veranstaltung belegen kann, wird durch die Verknüpfung der Veranstaltungen mit den definierten Zielgruppen festgelegt.

Veranstaltungen am HPI werden zudem von unterschiedlichen Lehrstühlen angeboten. So wird die Veranstaltung „Business Process Management II“ zum Beispiel vom Lehrstuhl von Prof. Weske angeboten und die Veranstaltung „Trends und Konzepte in der Softwareindustrie“ vom Lehrstuhl von Prof. Plattner. Die einzelnen Lehrstühle werden im Datenmodell durch die Entität *Chair* repräsentiert. Durch die Verknüpfung einer Lehrveranstaltung mit einem Lehrstuhl wird diese dem Lehrstuhl zugewiesen.

Mitarbeiter, die zu bewertende Veranstaltungen am HPI betreuen, werden im Datenmodell, wie bereits angedeutet, durch die Entität *Staff* repräsentiert. Es können grundsätzlich beliebig viele Mitarbeiter angelegt werden. Für jeden Mitarbeiter kann ein Name, ein Loginname und eine Foto gespeichert werden. Die Speicherung des Loginnamens ist notwendig, damit Dozenten Zugriff auf den Dozentenbereich im Evaluierungssystem haben und die von ihnen durchgeführten Veranstaltungen erkannt und angezeigt werden können. Das Foto bzw. der Verweis auf das Foto in Form einer URL wird gespeichert, um den Studenten bei einer Veranstaltungsbewertung ein Foto anzeigen zu können. Jeder Mitarbeiter kann einem bestimmten Lehrstuhl zugeordnet werden. Dafür muss eine Verknüpfung zwischen Lehrstuhl und Mitarbeiter hergestellt werden. Jeder Lehrstuhl hat zudem eine Person der diesen Lehrstuhl leitet, wie zum Beispiel Prof. Weske der den BPT-Lehrstuhl am HPI leitet. Diese Beziehung kann durch die zweite im Datenmodell dargestellte Verknüpfung hergestellt werden.

Verwaltung von Fragebögen

EvaJ ermöglicht das Anlegen eines digitalen Fragebogentemplates pro durchgeführter Evaluierung. Das Fragebogentemplate muss dabei nur einmal initial angelegt werden, wenn noch keine Evaluierungen durchgeführt wurden. Ansonsten wird für jede neu angelegte Evaluierung, das Fragebogentemplate der letzten Evaluierung also Vorlage kopiert. Aus dem Fragebogentemplate werden dann für die jeweiligen, zu bewertenden Veranstaltung entsprechenden Fragebögen generiert. Fragebogentemplates werden im Datenmodell durch die Entität *QuestionnaireTemplate* repräsentiert.

Jedes Fragebogentemplate besteht aus einer beliebigen Menge von Themenbereichstemplates. Themenbereichstemplates werden im Datenmodell durch die Entität *TopicTemplate* repräsentiert. Jedes Themenbereichstemplate hat einen deutschen und englischen Namen wie zum Beispiel „Fragen zur Vorlesung“ oder „Questions about the Lecture“. Themenbereichstemplates können zudem mit beliebigen Veranstaltungstypen verknüpft werden. Die Verknüpfung des Themenbereichstemplates mit dem Bezeichner „Fragen zur Vorlesung“ mit dem Veranstaltungstyp „Vorlesung“ bewirkt beim Generieren der Fragebögen für alle Vorlesungen, dass ein Themenblock mit der Überschrift „Fragen zur Vorlesung“ und entsprechenden Fragen im Fragebogen erscheint. Um Fragen nicht nur zu einer Veranstaltung sondern auch zu den Mitarbeitern stellen zu können, die diese Veranstaltung durchgeführt haben, kann das Attribut *Per_person* auf true gesetzt werden. Dies bewirkt, dass im generierten Fragebogen pro eingetragenem Mitarbeiter ein Themenblock mit entsprechenden Fragen zur Person angezeigt wird. Die Reihenfolge in der die Themenblöcke im generierten Fragebogen angezeigt werden, kann letztlich durch das Setzen einer entsprechenden Zahl für das Attribut *index* erreicht werden (0 – forderste Position, n-1 – vorletzte Position, n – letzte Position).

Jedem Themenbereich kann eine beliebige Anzahl von Fragetemplates zugewiesen werden. Jedes Fragetemplate steht für eine Frage die im generierten Fragebogen erscheint. Fragetemplates werden im Datenmodell durch die Entität *QuestionTemplate* repräsentiert. Jedes Fragetemplate hat einen deutschen und englischen Text, wie zum Beispiel „Wie fanden Sie den Inhalt der Vorlesung?“. Zudem kann der Antworttyp festgelegt werden. Momentan werden drei Typen unterstützt: „nur Text“, „Bewertung von 1-5“ und „Freitextantwort“. Zusätzlich kann auch die Gewichtung der Antworten bzgl. der Gesamtauswertung festgelegt werden.

Wird als Antworttyp die „Bewertung von 1-5“ festgelegt muss zusätzlich eine Verknüpfung zu einer bestimmten Antwortbeschreibung hergestellt werden. Antwortbeschreibungen werden im Datenmodell durch die Entität *AnswerDescription* repräsentiert. In einer Antwortbeschreibung werden die Texte für die jeweiligen Werte zwischen 1 bis

5 in deutscher und englischer Sprache hinterlegt, wie zum Beispiel 1, sehr gut bis 5, sehr schlecht oder 1, angemessen bis 5, nicht angemessen. Über die Antwortbeschreibungen können also individuelle Beschriftungen für bestimmte Fragen festgelegt werden, die über die für einen Bewerter wenig intuitiven Werte 1 bis 5 hinausgehen.

Zusätzlich zu dem generischen Fragebogentemplate aus dem die veranstaltungsspezifischen Fragebögen generiert werden, können auch veranstaltungsspezifische Fragen festgelegt werden. Diese erscheinen dann nur auf dem Fragebogen zur entsprechenden Veranstaltung. Diese Fragen werden im Datenmodell durch die Entität *AdditionalQuestionTemplate* repräsentiert. Wie beim Fragetemplate kann auch hier ein Text festgelegt werden, aus Gründen der Einfachheit allerdings nur in deutscher Sprache. Zudem ist der Antworttyp immer Freitext, weil sonst das Bewertungsergebnis durch bestimmte Fragen verwischt werden könnte. Über das Attribut *index* kann, wie im Fragetemplate, die Reihenfolge festgelegt werden, in der die Fragen im generierten Fragebogen erscheinen.

Verwaltung der Veranstaltungsbelegungen der Studenten

EvaJ erlaubt das Verwalten einer beliebigen Anzahl von Studenten und deren Belegungen. Studenten werden im Datenmodell durch die Entität *Student* repräsentiert. Für jeden Studenten wird nur der Loginname also zum Beispiel *martin.breest* gespeichert. Dies ist notwendig, damit für den Studenten die jeweiligen Belegungen angezeigt werden können.

Jeder Student kann eine beliebige Anzahl von Veranstaltungen belegen und dementsprechend evaluieren. Belegungen werden im Datenmodell durch die Entität *Enrollment* repräsentiert. Eine Belegung verknüpft genau einen verwalteten Studenten mit einer Lehrveranstaltung einer Evaluierung. Für jede Belegung kann zudem festgelegt werden, ob der Student bereits abgestimmt hat oder nicht. Ist das Attribut *AlreadyVoted* auf *true* gesetzt, kann der Student nicht mehr für die Veranstaltung abstimmen. Wichtig ist, dass es keinerlei Beziehung zwischen einer Belegung oder einem Studenten und den durchgeführten Evaluierungen gibt. Die Abstimmung in EvaJ ist also vollständig anonym. Auch das Datenmodell stellt dies sicher.

Jeder Student gehört zudem zu einer bestimmten Kategorie. Am HPI wird zwischen Bachelor- und Masterstudenten unterschieden. Studentenkategorien werden durch die Entität *StudentCategory* repräsentiert.

Durch die Verknüpfung eines Studenten mit einer bestimmten Studentenkategorie wird dieser entweder den Master- oder den Bachelorstudenten zugeordnet.

Verwaltung der einzelnen Bewertungen

EvaJ erlaubt das Verwalten beliebiger Veranstaltungsbewertungen. Wichtig ist, dass die Bewertungen unabhängig vom Bewerter gespeichert werden. Anonymität für den Bewerter wird also schon durch das Datenmodell sichergestellt. Bewertungen werden im Datenmodell durch die Entität *Assessment* repräsentiert.

Um eine Bewertung für eine Veranstaltung zu erstellen muss diese mit der entsprechenden Veranstaltung verknüpft werden. Es können beliebig viele Bewertungen pro Veranstaltung abgespeichert werden.

Bewertungen können zudem einer bestimmten Studentenkategorie zugewiesen werden. Dies ermöglicht beim Erstellen der Reports für eine Evaluierung das Aggregieren der Veranstaltungsbewertungen nach Bachelor- und Masterveranstaltungen.

Jede Bewertung besteht aus einer Menge von Antworten auf die im generierten Fragebogen gestellten Fragen. Antworten werden im Datenmodell durch die Entität *Answer* repräsentiert. Jede Antwort ist mit genau einem Fragentemplate verknüpft. Da bei den Fragen zwischen Fragen zu einer Veranstaltung und Fragen zu einem Mitarbeiter unterschieden wird, kann eine Antwort zudem mit dem Mitarbeiter zu dem die Frage gestellt wurde verknüpft werden. Jede Antwort enthält für Fragen des Typs „Bewertung von 1-5“ den entsprechenden Wert zwischen 1 und 5. Zudem kann für Fragen des Typs „Bewertung von 1-5“ und „Freitext“ ein Kommentar gespeichert werden. Da in EvaJ Kommentare zensiert werden können, kann zusätzlich der Status der Zensur, also „nicht zensiert“, „zensiert“ und „nicht veröffentlichen“, und der zensierte Kommentar gespeichert werden.

Antworten zu veranstaltungsspezifischen Fragen werden in EvaJ separat gespeichert. Diese Antworten werden durch die Entität *AdditionalAnswer* repräsentiert. Jede dieser Antworten ist mit einer veranstaltungsspezifischen Frage verknüpft. Die Antwort enthält nur einen Kommentar da veranstaltungsspezifische Fragen, wie bereits erwähnt, nur vom Typ Freitext sein können.

Verwaltung der aggregierten Veranstaltungsergebnisse

EvaJ aggregiert die Einzelbewertungen für eine Veranstaltung, sobald diese vom Fachschaftsrat veröffentlicht wurde. Die aggregierten Einzelbewertungen werden als Umfrageergebnisse gespeichert. Umfrageergebnisse werden im Datenmodell durch die Entität *SurveyResult* repräsentiert. Für jedes Umfrageergebnis wird die Durchschnittsnote, die sich aus den Antworten auf alle Fragen des Typs „Bewertung von 1-5“ ergibt, die Varianz, die Anzahl der Belegungen und die Anzahl der abgegebenen Stimmen gespeichert.

Jedes Umfrageergebnis besteht zudem aus einer Menge von Themenbereichsergebnissen. Themenbereichsergebnisse werden im Datenmodell durch die Entität *TopicResult* repräsentiert. Für jedes Themenbereichsergebnis wird der Durchschnittswert, der sich aus den Antworten auf alle Fragen des Typs „Bewertung von 1-5“ des Themenbereichs ergibt, und die Varianz gespeichert. Zudem sind die Themenbereichsergebnisse mit dem jeweiligen Themenbereichstemplate aus dem generischen Fragebogen verknüpft. Handelt es sich dabei um ein Themenbereichstemplate das für Mitarbeiter und nicht für Veranstaltungen eingestellt ist, wird zudem eine Verknüpfung zum bewerteten Mitarbeiter hergestellt. Diese Verknüpfung ist notwendig, um zu wissen zu welchem Mitarbeiter die aggregierten Themenbereichsergebnisse gehören.

Jedes Themenbereichsergebnis besteht zudem aus einer beliebigen Menge von Antwortergebnissen. Antwortergebnisse werden im Datenmodell durch die Entität *AnswerResult* dargestellt. Ein Antwortergebnis ist mit einem Fragentemplate aus dem generischen Fragebogen verknüpft und aggregiert die Bewertungen aller Studenten zu einer Frage für eine bestimmte Veranstaltung. Für jedes Antwortergebnis wird der Durchschnittswert aller Bewertungen zu der entsprechende Frage, die Varianz, und die Anzahl der abgegebenen Stimmen pro Note für die Noten zwischen 1 bis 5 gespeichert. Diese Daten werden später benötigt um Histogramme anzuzeigen.

Verwaltung allgemeiner Aspekte

EvaJ erlaubt es auch bestimmte Konfigurationen in der Datenbank abzulegen. Momentan werden nur die Email-Texte und die Teilnehmer der Auslosung pro Evaluierung gespeichert.

1.3 Sicherheitsaspekte

Ziel dieses Abschnitts ist es, das Sicherheitskonzept von EvaJ vorzustellen. Dabei sind Sicherheitsaspekte besonders an zwei Stellen relevant: Zum einen in der Authentifizierung und Autorisierung des Benutzers, zum anderen bei der Kommunikation zwischen Frontend und Backend. Diese beiden Stellen sollen im Folgenden beleuchtet werden.

Zugriffsschutz via Kerberos

Das EvaJ Frontend besteht aus zwei Webservern. Der vorgelagerte Apache Webserver nimmt zunächst alle eingehenden Anfragen entgegen. Wenn JSP- oder JSF-Inhalte vom Browser nachgefragt werden, leitet der Apache Webserver an einen Tomcat Server weiter. Das EvaJ Backend besteht aus einem Web Application Server für Java Anwendungen. In Frage kommende Application Server sind JBoss und Tomcat. Im Folgenden wird zunächst der technische Ablauf bei der Anmeldung des Benutzers beschrieben. Der Benutzer ruft über den Webbrowser seiner Wahl die URL von EvaJ auf. Dabei wird die Anfrage zunächst vom Apache Webserver entgegen genommen. Bereits an dieser Stelle wird die Kommunikation zwischen dem Browser und dem Apache Webserver verschlüsselt. Der Apache Webserver wird mit dem Modul `mod_auth_kerb` betrieben. Dieses Modul erlaubt es, Ressourcen des Apache Servers mit Kerberos zu schützen. Im Falle von EvaJ werden dabei zwei Fälle unterschieden: Entweder befindet sich der Benutzer im HPI und ist auf einem Windows-Rechner eingeloggt, oder nicht.

Betrachten wir zunächst den ersten Fall, dargestellt in Abbildung 4: Wenn der Browser die URL von EvaJ beim Webserver anfordert, wird dieser einen HTTP-Header zurücksenden, der den Browser zur Authentifizierung mittels Kerberos über das Negotiate-Protokoll veranlasst. Dazu greift der Browser (zurzeit nur Internet Explorer oder Mozilla / Firefox mit der entsprechenden Konfiguration) auf die Anmeldung des Benutzers am Windows-System zurück. Bei der Anmeldung am Windows-System läuft im Hintergrund bereits eine Authentifizierung über die Kerberos-Implementierung von Microsoft ab, die gegen das Key Distribution Center (KDC) gefahren wird. Das KDC gehört zum Active Directory (AD) des Windows Domain-Controllers. Bei erfolgreicher Anmeldung am Windows-System wird ein Ticket Granting Ticket (TGT) im Ticket Cache der Local Security Authority (LSA) des Windows-Systems abgelegt. Das TGT wird bei beim Aufrufen der EvaJ URL an den Apache Webserver übergeben. Dadurch wird eine Single-Sign-On Verhaltensweise des Systems EvaJ realisiert.

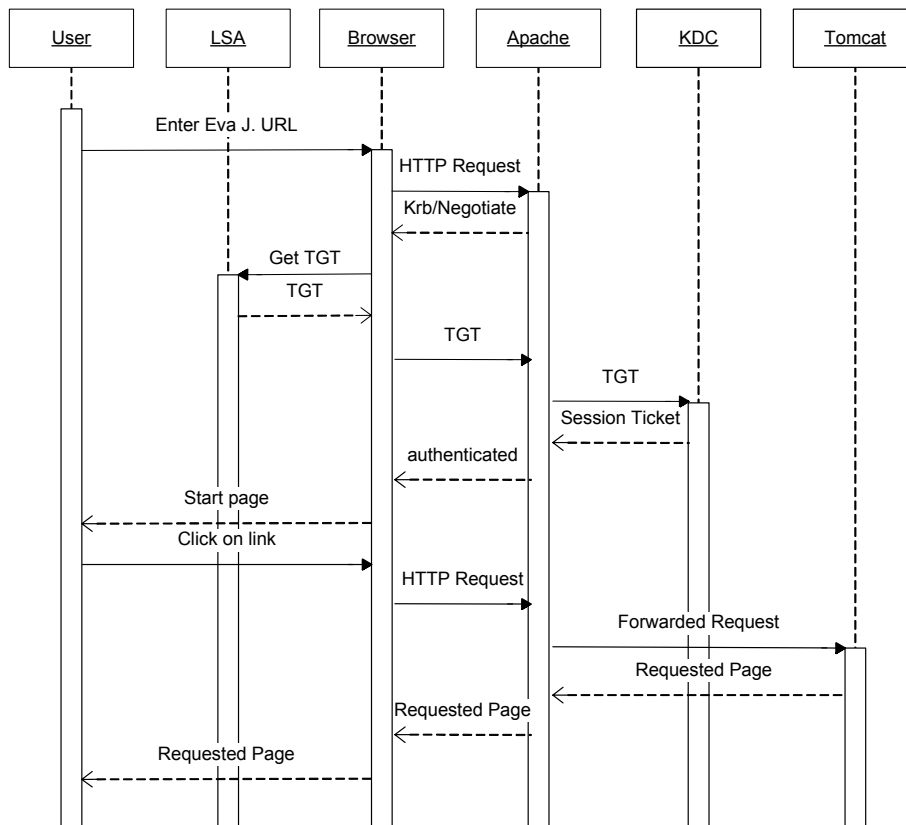


Abbildung 4: Kerberos innerhalb des HPI.

Wenn der Benutzer von außerhalb des HPI auf EvaJ zugreifen möchte, stellt sich der Ablauf geringfügig anders da. Der Ablauf für diesen Fall wird in Abbildung 5 dargestellt. Außerhalb des HPI können die Browser die Aufforderung des Apache Webservers zur Authentifizierung mittels Negotiate nicht verarbeiten. Selbst wenn ein TGT im Ticket Cache des Client-Rechners vorgefunden wird, passt es nicht mit dem Kerberos-Realm innerhalb des HPIs zusammen. In diesem Fall, aber auch wenn Negotiate vom verwendeten Browser nicht unterstützt wird, wird der Browser versuchen, eine Authentifizierung über HTTP BASIC-Auth einzuleiten. Dabei wird der Benutzer aufgefordert, einen gültigen Benutzernamen mit Passwort einzugeben. Diese Daten werden im BASE64-Format zum Apache Webserver übertragen. Da es sehr einfach ist, BASE64-verschlüsselte Passwörter in Klartext umzuwandeln, muss die Verbindung zum Apache Webserver mit SSL verschlüsselt werden. Der Apache Webserver wird dann mit diesen Zugangsdaten beim KDC ein Ticket anfordern. Ist dies erfolgreich, gilt der Benutzer als authentifiziert.

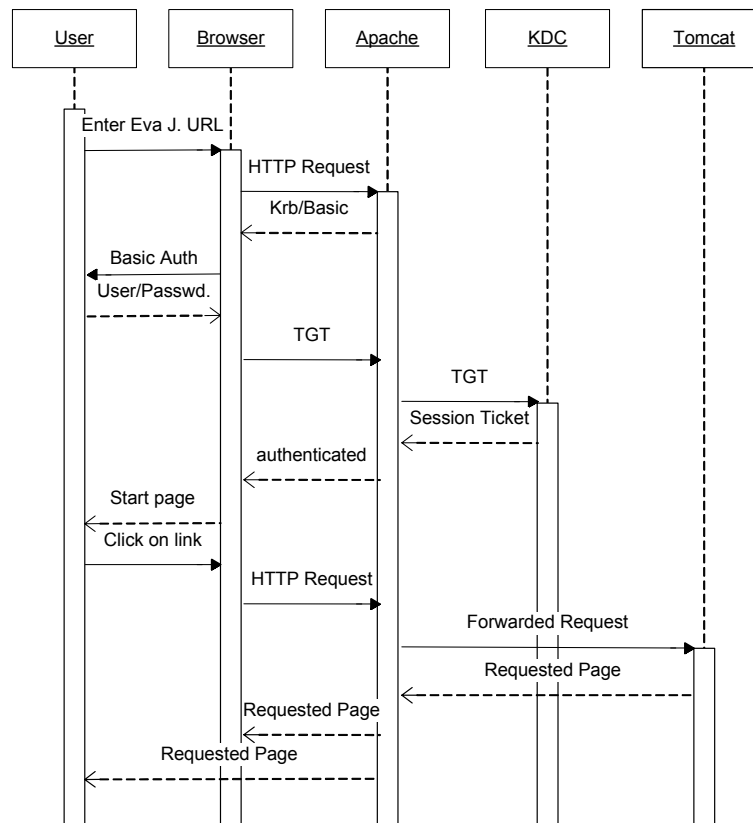


Abbildung 5: Kerberos außerhalb des HPI.

Kommunikation zwischen Frontend und Backend

Nach erfolgreicher Authentifizierung muss geprüft werden, für welche Aktionen innerhalb EvaJ der angemeldete Benutzer autorisiert ist. Die oben beschriebene Authentifizierung läuft im Frontend ab, die Autorisierung dagegen im Backend. Damit die Autorisierung durchgeführt werden kann, wird der Benutzername des authentifizierten Benutzers an das Backend weitergegeben. Dazu wird eine Login-Service-Methode aufgerufen, die dem Benutzernamen nach einer Anfrage gegen den OpenLDAP-Server die entsprechenden Rollen zuweist. Im Backend wird ein so genanntes UserContext-Objekt erzeugt, das Benutzername und Rollen vorhält. Der UserContext wird an das Frontend zurückgegeben, damit es von diesem wiederum mit jedem Serviceaufruf an das Backend als Parameter übergeben werden kann.

Die Möglichkeit, dass böswillige Personen mittels Reverse-Engineering eigene UserContext-Objekte erzeugen und damit in die Lage versetzt werden, beliebige Services im Kontext des vorgetäuschten Benutzers im Backend aufzurufen, muss ausgeschlossen werden. Deshalb wird die Kommunikation zwischen Frontend und Backend ebenfalls auf Transportebene mit SSL abgesichert. Entscheidend dabei ist, dass das Backend derart konfiguriert ist, dass es nur vom EvaJ-Frontend aufrufbar ist. Eine eindeutige Identifizierung des Frontends gegenüber dem Backend wird durch das X.509 Zertifikat des Frontends möglich, welches für die SSL-Verbindung verwendet wird.

2 Projektdurchführung

Während der Entwicklung von EvaJ war das Projektteam einen Großteil der Zeit geteilt. Vier der Teammitglieder waren vor Ort in Potsdam, drei weitere entwickelten im Sauerland. Diese räumliche Trennung verursachte einigen Mehraufwand an Koordination und Projektleitung. Der folgende Abschnitt beschreibt unser Vorgehen, mit dem wir dieser Herausforderung begegnet sind. Dabei wird auf die Verteilung der einzelnen Zuständigkeitsbereiche und auf unser Vorgehensmodell eingegangen, sowie auf die Werkzeuge, die uns bei der entfernten Projektkoordination unterstützt haben.

2.1 Projektstruktur

Eine wichtige Aufgabe der Projektleitung am Anfang der Entwicklung eines Softwareprojektes besteht darin, Subteams zu bilden, die möglichst produktiv zusammenarbeiten. Um die Effizienz zu maximieren, musste in unserem Projekt vor allem darauf geachtet werden, dass die Entwicklerrollen mit dem höchsten Kommunikationsaufwand durch den kürzesten Kommunikationsweg verbunden waren. Es galt also Projektgruppen gemäß ihrer Erfahrung sowie ihres Aufenthaltsortes zusammenzustellen.

Von Anfang an wurde in der Veranstaltung Trends und Konzepte die Bedeutung der Benutzbarkeit der entwickelten Systeme hervorgehoben. Die in der ersten Phase entworfenen Vorschläge für die Benutzerschnittstelle fielen dementsprechend sehr detailliert und reich an Funktionalität aus. Der größte Aufwand entstand daher bei der Entwicklung des Frontends, weswegen diese Aufgabe mit einem Subteam aus vier Personen besetzt wurde. Die einzelnen Webseiten besitzen untereinander nur lose Abhängigkeiten. Nach deren Identifikation sowie einer Priorisierung von Aufgaben konnte die Arbeit hier gut aufgeteilt sowie Zuständigkeiten formuliert werden.

Das zweite Subteam übernahm die Aufgabe, das Backend zu entwickeln. Da bei der Implementierung eines so umfangreichen Datenmodells viele Abhängigkeiten beachtet werden müssen und so ein äußerst hoher Kommunikationsaufwand entsteht, war hier eine direkte Zusammenarbeit vor Ort unausweichlich. Dieses Team übernahm auch einen Großteil des Schnittstellendesigns, da dieses zwar durch Analyse des Prototyps abgeleitet wurde, jedoch Hand in Hand mit dem Datenbankdesign erstellt werden sollte. Im Aufgabenbereich dieser Gruppe lag auch die Anbindung an bestehende Infrastruktur zur Benutzer- und Rechteverwaltung, namentlich an Kerberos und LDAP.

Die scharfe Trennung der Aufgabenverteilung behielt bis zum Ende der Implementierungs- und Testphasen Bestand. Erst in der Einsatzphase wurden neue Rollen zur Systemadministration, Nutzerbetreuung und Fehlerbeseitigung vergeben.

2.2 Vorgehensmodell

Durch die räumliche Trennung der Projektmitglieder war es von wesentlicher Bedeutung, zunächst Schnittstellen zwischen den einzelnen Aufgabenbereichen zu definieren. Wir entschieden uns daher für ein klassisches Vorgehensmodell der Softwareentwicklung, also eine Unterteilung des Entwicklungsprozesses in die Phasen Analyse, Design, Implementierung, Test und Einsatz. Variiert wurde dieser Entwicklungsprozess durch das umfangreich erstellte User Centric Design, das bereits in sehr frühen Phasen einen deutlichen Fokus auf die Benutzungsschnittstelle vorgab.

Analyse

Die Analysephase wurde bereits im ersten Teil der Veranstaltung abgeschlossen. Anstelle der gewohnten Pflicht- und Lastenhefte oder textueller Spezifikationsdatenbanken entstand hier ein Prototyp sowie ein Dokument, das die Benutzung und Funktionalität der einzelnen Webseiten beschreibt.

Design

Die Designphase diente im Wesentlichen dazu, die Schnittstelle zwischen den beiden Teilteams zu definieren. Erst danach konnten die Subteams das Feindesign ihrer Unterkomponenten unabhängig von einander durchführen und diese dann implementieren. Beim Design der Schnittstelle konnte sehr systematisch vorgegangen werden. Zunächst wurde Schritt für Schritt die Dokumentation der Benutzerschnittstelle durchgegangen und an jeder einzelnen Seite annotiert, welche Daten zur ihrer Generierung nötig sind.

Sobald alle nötigen Daten bekannt waren, konnte ein Datenmodell entwickelt werden, das die Beziehungen all dieser Daten untereinander darstellt. Das hierbei entstandene Datenmodell wird im Abschnitt "Das Datenmodell von EvaJ" auf Seite 8 beschrieben. Ausgehend vom Datenmodell konnte zum Einen das Design der Datenbank, zum Anderen das der Schnittstelle definiert werden. Hierbei besteht die Schnittstelle aus zwei Teilen. Der erste besteht aus Klassen, so genannten Value Objects, die die einzelnen Datensätze des Datenmodells sowie ihre Beziehungen untereinander nachbilden, dabei jedoch keinerlei Anwendungslogik enthalten.

Der zweite Teil sind eine Menge sogenannter Services, die Methoden bündeln, mit denen das Frontend Value Objects vom Backend erfragen kann, beziehungsweise aktualisierte Daten zurückschicken kann. Diese Services werden dann vom Backend implementiert. Für die Bestimmung der Services wurden die Webseiten in Gruppen zusammengefasst, die ähnliche oder gleiche Daten benötigen. Ein Service fasst dann die Daten zusammen, die eine Gruppe von Webseiten benötigt. So gibt es beispielsweise einen Service, der die Funktionalität der Fragebogen anbietet und einen, der alle Daten bietet, die benötigt werden, um die Auswertung von Evaluierungen anzuzeigen.

Um eine unabhängige Entwicklung der beiden Teilteams zu ermöglichen, entstand die Implementierung der Schnittstelle noch in der Designphase. Dazu mussten die Value Objects als Klassen, die Referenzen aufeinander halten, und die Services als Interfaces formuliert werden, die vom Backend implementiert werden können. Um die Trennung von Frontend und Backend deutlich zu machen, wurde die Schnittstelle in einem eigenen Eclipse-Projekt definiert. Dadurch können Frontend und Backend unabhängig von einander ausgetauscht werden und auch die Kommunikation zwischen den beiden Teilprojekten variabel gestaltet werden. So gibt es beispielsweise die Möglichkeit statt einfacher lokaler Methodenaufrufe die Schichten durch Webservices zu trennen.

Implementierung

Da die Implementierung des Backends und des Frontends nebenläufig verlief, werden diese Teilprojekte hier getrennt betrachtet.

Frontend

Im Frontendprojekt sind die Webseiten enthalten, die der Benutzer des Systems später zu Gesicht bekommt. Diese können nur sinnvoll entwickelt werden, wenn der Programmierer über Beispieldaten verfügt. Da das Frontend unabhängig vom aktuellen Entwicklungsstatus des Backends gebaut wurde, wurde zunächst ein so genanntes Dum-

my-Backend erstellt, das zwar die selbe Schnittstelle wie das Backendprojekt implementiert, die Funktionalität jedoch nur mit ausreichend sinnvollen Testdaten simuliert, um so eine parallele Entwicklung zu ermöglichen.

Das Frontend selber wurde streng nach dem Model-View-Controller (MVC) Pattern entwickelt. Das Model wird gebildet durch die vom Backend gefüllten Value Objects aus dem Interface Projekt, sowie von einigen, dem Frontend eigenen Wrapperklassen. JavaServerFaces wurden genutzt, um die Views zu erstellen, die Controllerfunktionalität übernehmen sogenannte Backing-Beans. Siehe Abschnitt "Die Architektur von EvaJ" auf Seite 5 und "Erfahrungen zur Entwicklung" auf Seite 20. Nach der Entwicklung jeder Seite folgte ihr Oberflächendesign, also das „Styling“ der Webseite. Dabei wurde zumeist darauf geachtet, dem Prototyp möglichst nahe zu kommen, an einigen Stellen flossen jedoch auch Verbesserungsideen ein. Großer Wert wurde darauf gelegt, das Oberflächendesign zu externalisieren, um so jederzeit einfach und unkompliziert das verwendete „Look and Feel“ durch ein neues austauschen zu können

Backend

Im Backend wurde zunächst das Datenmodell aus der Designphase mit einer Datenbank umgesetzt. Im Anschluss konnte die Anwendungslogik implementiert werden, also der Teil der Anwendung, der die im Frontend benötigten Daten aus der Datenbank beschafft sowie die gewünschten Änderungen auf der Datenbank durchführt. Das Backend implementiert damit das Schnittstellenprojekt und „übersetzt“ die dortigen Methodenaufrufe in Aktionen auf der Datenbank.

Sobald einzelne Komponenten des Backends fertiggestellt waren, konnten die entsprechenden Teile im Dummy-Backend schrittweise durch die nun lauffähigen Systeme ersetzt werden. Dafür wurden umfangreich Testdaten in die Datenbank eingespielt. Die Anbindung an Kerberos/LDAP konnte durch den Kommunikationsaufwand mit dem HPI erst spät fertiggestellt werden. Deswegen, und um die verschiedenen Rollen wie Dozent, Fachschaftsrat oder Student testen zu können, wurde eine Dummy-Nutzerverwaltung entwickelt. Diese ist im Produktivbetrieb ausgeschaltet, lässt im Testbetrieb jedoch Testbenutzer zu, die jeweils eine Rolle emulieren können.

Test

Von Beginn an wurde bei EvaJ das Ziel gesteckt, das System für die Praxis einsetzbar und stabil zu gestalten. Um die Einsetzbarkeit des Evaluierungssystems EvaJ garantieren zu können, wurde viel Arbeit auf das Testen der Anwendung verwendet. So wurden Skripte geschrieben, die umfangreiche Testdaten wiederholbar aufbauen. Für die verschiedenen Services im Backend bestehen Testklassen, die per Unittests das Backend auf Korrektheit überprüfen. Und das Frontend wurde durch zeitintensive Benutzerinteraktion getestet.

Einsatz

Um sich nicht auf die gängigen Methoden der Softwarebranche beim Testen verlassen zu müssen, sollte EvaJ noch vor ihrem geplanten Produktiveinsatz einen Probelauf absolvieren. In diesem äußerst erfolgreichen Testeinsatz konnten einige Ideen gefunden werden, die das System für ihren letztendlichen Einsatz noch verbesserten. Für die Einsatzphase wurden die Rollen der Entwickler neu verteilt in Systemadministratoren, in Supporter, die die Benutzer bei Fragen und Problemen unterstützen und in die Entwickler, die gefundene Fehler im System beseitigen und neue Änderungswünsche einbauen. Diese Phase begann äußerst arbeitsintensiv, der Aufwand flachte jedoch ziemlich schnell ab. Mehr zu den Erfahrungen, die in dieser Phase bisher gesammelt wurden, werden im Abschnitt "Er-

fahrungen zur Entwicklung” auf Seite 20 besprochen.

2.3 Herausforderungen Fernarbeit

Bereits in Teams von sieben Leuten können Probleme bei der Synchronisation von Wissen und der genauen Verteilung von Zuständigkeiten auftreten. Um die Zusammenarbeit im Team zu optimieren wurden verschiedene Mechanismen verwendet. Der erste Schritt in diese Richtung war eine klare Unterteilung in zwei unabhängige Teilteams und die dafür notwendige Definition der Schnittstelle zwischen Frontend und Backend. Anhand der Schnittstelle konnte das Backend-Team vereinbaren, wer für welche Implementierungsaufgaben zuständig ist. Das Frontend-Team kategorisierte alle im Prototyp beschriebenen Seiten nach Priorität und Aufwand, um so allen Mitgliedern gleiche Aufgaben zuteilen zu können.

Um regelmäßig über den Fortschritt des Projektes informiert zu sein, gab es regelmäßige Meetings, in denen Statusberichte vorgetragen, das weitere Vorgehen geplant, und neue Aufgaben verteilt wurden. Diese Treffen wurden meist durch Telefonkonferenzen realisiert. Durch eigens eingerichtete Mailinglisten und Chatprogramme waren die verschiedenen Entwickler untereinander gegenseitig leicht erreichbar.

Bei einem solch umfangreichen Projekt ist es unausweichlich, dass verschiedene Personen parallel auf einigen gleichen zentralen Dateien arbeiten müssen. Zur Synchronisierung und Versionierung dieser Daten sowie des kompletten Codes wurden mächtige Tools wie Subversion/SVN, Diff und Merge genutzt. Das komplette System wurde unter Eclipse entwickelt. Diese Umgebung bietet ein Werkzeug, mit dem umfangreiche TODO-Listen bequem verwaltet werden können. Fehlender Code wurde so immer als TODO markiert, um immer klar sehen zu können, welche Aufgaben noch erledigt werden müssen.

Des Weiteren wurden verschiedene Konventionen fixiert. So galt beispielsweise die Regel, dass Änderungen an vorher vereinbarten Schnittstellen nach Möglichkeit vermieden werden sollen. Falls sie doch nötig werden sollten, so mussten diese über die Mailingliste an alle Projektteilnehmer propagiert werden. Auch wurde Dokumentation von Code sehr wichtig genommen. Code, der von anderen weiterverwendet wird, musste beschrieben werden, um die Benutzung zu erleichtern. Im gesamten Projekt gibt es dafür Kommentare, im Schnittstellenprojekt wurde JavaDoc verwendet, um das Austauschen der Komponenten gegen Neuentwicklungen zu vereinfachen.

3 Erfahrungen zur Entwicklung

Im Folgenden beschreiben wir die Erfahrungen, die wir während des Entwicklungsprozesses mit den verwendeten Technologien und Werkzeugen gemacht haben. Erwähnenswert hierbei ist, dass wir uns aufgrund der (potentiellen) späteren freien Verwendung von EvaJ als Produktivsystem auf OpenSource-Werkzeuge festgelegt haben. Lediglich mit *Exadel Studio* haben wir ein nicht freies, dafür aber auf die *OpenSource*-Entwicklungsumgebung Eclipse basierendes und in der Basisversion kostenloses Werkzeug verwendet. Da der Quelltext vom verwendeten Werkzeug vollständig unabhängig ist, sollte das für eine spätere Weiterentwicklung durch Dritte keine Probleme bereiten.

3.1 Erfahrungen mit den verwendeten Technologien

Zur Entwicklung haben wir weit verbreitete *OpenSource*-Standards verwendet. Während wir das Backend mit Hibernate und Spring realisiert haben, kamen bei der Entwicklung des Frontends *Java ServerFaces* (JSF) zum Zuge.

JSF

Java ServerFaces ist ein komponentenbasierter Framework-Standard zur Entwicklung von Webanwendungen. Anstelle der traditionellen Erstellung von um Programmstrukturen erweiterten HTML-Seiten, lassen sich mit JSF Web-Oberflächen aus Komponenten zusammenstellen. Die Entwicklung der Views geschieht so auf einem hohen Abstraktionsniveau, ist eher deklarativ als imperativ und entspricht so mehr der traditionellen Beschreibung von GUI-Oberflächen von Desktopanwendungen. Komponenten sind in sog. Tag-Libraries verfügbar. Für die Oberflächen von EvaJ haben wir die *Tomahawk*-Komponenten der Apache *MyFaces*-Referenzimplementierung verwendet.

In der Praxis hat sich JSF durchaus bewährt. Das Konzept von JSF gut durchdacht und schnell verständlich. Die Einarbeitungszeit in die Technologie war eher gering. Dennoch wurden einige *Best Practices* erst nach einigen Wochen Arbeitszeit ersichtlich. Durch den deklarativen Ansatz konnten die einzelnen Seiten in gut verständlichen und kompakten Quelltext umgesetzt werden. Die *Tomahawk*-Komponenten bieten einen großen Umfang an Standardfunktionalität. Leider konnte diese an machen Stellen aber nicht wie erwartet bereitgestellt werden. So können mit der von uns verwendeten Version z.B. keine Kalender zur einfachen Datumsauswahl angezeigt werden, da die generierten *JavaScript*-Funktionen fehlerhaft sind.

Probleme im Zusammenhang mit unserem nebenläufigem Arbeiten mit Hilfe von *Subversion* haben die zentralen Konfigurationsdateien mit sich gebracht, da an ihnen vorgenommene Änderungen auch berücksichtigt werden müssen, wenn die entsprechende Funktionalität für den jeweils eigenen Entwicklungsbereich irrelevant ist.

Da nicht für alle Belange Komponenten zur Verfügung stehen (z.B. das Anzeigen von Graphen zur Auswertung der Evaluationsergebnisse), mussten wir entsprechende Funktionalität selbst implementieren. Dies geschah weitestgehend ohne Probleme. Für einige besondere Features wie dem *Akkordeon* zur Gruppierung der Frageblöcke auf dem Bewertungsbogen, haben wir die modernen Web 2.0 JavaScript Bibliotheken *OpenRico*, *script.aculo.us* und *Prototype* verwendet. Zur vollständigen Anpassung der Funktionalität mussten wir gelegentlich in den Code der meist unzureichend dokumentierten Bibliotheken selbst eingreifen, wozu sehr gute *JavaScript*-Kenntnisse erforderlich waren. Im Allgemeinen haben wir einen großen Teil unserer Zeit für die Programmierung in *JavaScript* verwendet.

3.2 Erfahrungen mit der Entwicklungsumgebung

Als Entwicklungsumgebung für das Frontend haben wir *Exadel Studio* verwendet. Dabei kamen sowohl die zeitlich beschränkte Testversion der kommerziellen Pro- als auch die um einige Features beschränkte Standard-Version zum Einsatz. Beide Versionen sind kostenlos auf der Website von *Exadel Inc.* verfügbar.

Da *Exadel Studio* auf der beliebten und uns allen gut bekannten und geschätzten OpenSource-Entwicklungsumgebung *Eclipse* basiert, konnten wir uns sehr schnell einarbeiten. Für die Realisierung des Backends haben wir *Eclipse* ohne die *Exadel*-Erweiterungen verwendet. Für die Programmierung in Java bietet die Werkzeugunterstützung von *Eclipse* viele Vorteile, wie z.B. automatisches Importieren von Paketen, Sprung zur referenzierten Klassendeklaration, etc. Nicht zuletzt war auch die große Zahl an Erweiterungen, wie z.B. der in *Eclipse* integrierbare SVN-Client *Subclipse* für uns sehr hilfreich.

Die Unterstützung zur Erstellung von JSF durch *Exadel Studio* hat sich an einigen Stellen sehr bewährt. Insbesondere der semigrafische Editor zum Bearbeiten der zentralen XML-Konfigurationsdatei und die integrierte Möglichkeit der Verwaltung des mitgelieferten Tomcat-Servers waren von Hilfe. Mit dem Editor kann z.B. die Navigationsstruktur zwischen den verschiedenen Seiten durch „Zeichnen“ der Navigationspfade bestimmt werden. Im Vergleich zum Editieren der relativ großen und schlecht lesbaren XML-Datei bietet *Exadel Studio* hier eine große Hilfe. Hilfreich war auch, dass *Exadel Studio* die referenzierten Backing Beans „kennt“ und sowohl Attribute als auch die Inhalte der externalisierten Sprachdateien per Auto-vervollständigung zur Verfügung stehen.

Als weniger nützlich hat sich der grafische WYSIWYG-Editor zum Aufbau des Seiteninhalts und Layouts erwiesen. Einerseits war der von uns erstellte Inhalt meist zu dynamisch, als dass er in einem WYSIWYG-Editor darstellbar wäre, andererseits ist die Möglichkeit des graphischen Einfügens von Komponenten nicht wirklich intuitiv und problemlos verwendbar. Das Editieren des Quellcodes selbst jedoch ist sehr komfortabel möglich, insbesondere aufgrund der Autovervollständigung der zu Komponenten verfügbaren Attribute.

Während wir mit der Funktionalität von *Exadel Studio* weitestgehend sehr zufrieden waren, hat uns die schlechte Performance der Entwicklungsumgebung große Probleme bereitet. Da das Produkt vor allem vom Frontend-Team verwendet wurde, das sich zur Zeit der Entwicklungsphase nicht in Potsdam aufhalten konnte, mussten wir es auf unseren privaten Notebooks installieren und konnten nicht auf die leistungsstarken Workstations des EAA Labs zurückgreifen. So konnte z.B. ein Notebook mit 512 MB Arbeitsspeicher und 1,9 GHz Taktfrequenz nur unter großen Schwierigkeiten eingesetzt werden. Jeder Schritt, auch nur das einfache Editieren eines XML-Attributs erforderte teilweise minutenlange Wartezeiten, bis ein weiteres Eingreifen möglich war. Potentiell kleine Änderungen konnten, insbesondere wenn ein Neustart des Servers oder ein erneutes Kompilieren des gesamten Projektes erforderlich waren, so erst binnen mehrerer Stunden durchgeführt werden. Diese langsame Art des Arbeitens ist nicht nur auf die schlechte Performance der Entwicklungsumgebung zurückzuführen, sondern auch auf die Entwicklung von Web-Anwendungen in Java allgemein. Zur Überprüfung des Ergebnisses muss nach jeder Änderung das Kompilat auf dem Server bereitgestellt werden, bei Änderungen bestimmter referenzierter Klassen der gesamte Code neu kompiliert und der Server neu gestartet werden. Die Web-Anwendungsentwicklung mit Skriptsprachen wie *Ruby* sollte in diesem Punkt einen klaren Vorteil bereiten.

4 Erfahrungen zu EvaJ im Einsatz

Nachdem durch User-Centered Design die Anforderungen gesammelt wurden und erste Oberflächenprototypen vorlagen, in der Entwurfsphase Schnittstellendefinitionen und Datenmodellierung vorgenommen wurden, nach der Implementierungs- und Testphase die technische Realisierung vorlag, fehlten nun noch zwei wichtige Phasen im Softwarelebenszyklus: Betrieb und Wartung.

EvaJ hat die Ehre, als Produktivsystem am HPI eingesetzt zu werden. Dies stellt das System auf den Prüfstand. Nun stellt sich heraus, ob das System zuverlässig funktioniert, mit der Menge an Benutzern zurechtkommt und wie leicht sich Änderungen noch ins laufende System einarbeiten lassen.

Weitestgehend hat EvaJ halten können, was versprochen war. Nichtsdestotrotz sind wir auf eine Reihe von Herausforderungen gestoßen, die für weitere Evaluierungen angegangen werden müssen. Wir unterscheiden im Folgenden technische, organisatorische und funktionale Herausforderungen.

4.1 Technische Herausforderungen

Allgemein kann man sagen, dass sobald jemand erst einmal eingeloggt war, alles relativ problemlos funktioniert hat.

Folgende Fehlerquellen haben wir identifiziert:

- *Rechtschreibfehler* bei der Eingabe von Dozentennamen. Die Dozentennamen, die nicht per Belegungsimport in das System gelangen, werden per Hand auf der Lehrveranstaltungskonfigurationsseite eingegeben. Kleine Rechtschreibfehler haben in mehreren Fällen dazu geführt, dass sich die entsprechenden Dozenten nicht einloggen konnten.
- *Exotische Browser*. Während der Entwicklung wurden lediglich die Browser Firefox, Internet Explorer und Opera verwendet, um das System immer wieder zu testen. Es hat sich herausgestellt, dass zu einem (verschwindend geringen) Prozentsatz auch weniger verbreitete Browser verwendet wurden. Diese unterstützen z.T. SPNEGO nicht, sodass die Authentifizierung in diesen Fällen gescheitert ist.
- *Abweichungen Windows-Login vs. Email-Adresse*. Aufgrund der Länge mancher Windows-Loginnamen wurde diese für das HPI-Netz abgekürzt (z.B. „alexander.klimetschek“ => „alex.klimetschek“). Für die Email-Adressen wurden jedoch die kompletten Namen verwendet. Dies führt dazu, dass die im System bekannten Loginnamen nicht benutzt werden können, um Erinnerungsmails zu verschicken.

4.2 Organisatorische Herausforderungen

- Zu wenig Zeit zur Konfiguration der LVs + eigenen Fragen (min. 2 Wochen!)
- Wissen über alle Dozenten / Übungsleiter / etc. im FSR nicht vorhanden
- Fragen zu Seminaren / Projekten nicht definiert
- Bei besonderen Veranstaltungstypen (z.B. Sprechtraining) passen die existierenden Fragen nicht
- Fragen zum Fragebogen: Sollen die Antworten dem Dozenten zukommen?

4.3 Funktionale Herausforderungen

Es hat sich gezeigt, dass in der User-Centered Design Phase nicht alle wichtigen Anforderungen aufgenommen worden sind.

- *Zuordnung von Kommentaren zu Fragen.* Im Eva-Prototyp wurde nicht beachtet, dass Kommentare sowohl beim Zensieren als auch bei der Anzeige für die Dozenten den jeweiligen Fragen zugeordnet werden müssen. Eine weitere Frage ist, wer eigentlich die Kommentare zu „Fragebogen allgemein“ sehen soll. Die jeweiligen Dozenten einer Lehrveranstaltung dürfte es wohl kaum interessieren.
- *Autorisierung bei Lehrveranstaltungsconfigurationen.* Im Prototyp war vorgesehen, dass nur diejenigen Lehrpersonen eine Veranstaltung konfigurieren dürfen, die auch später von Studenten bewertet werden können. Da viele Veranstaltungen allein durch wissenschaftliche Mitarbeiter durchgeführt werden, haben meistens die importierten Zuordnungen von Dozenten zu Lehrveranstaltungen nicht gestimmt. Durch das aktuelle Design der Anwendung ist es nun der Fall, dass die offiziellen Anbieter eines Kurses die Kommentare nicht anschauen können, da sie nicht zu den bewerteten Lehrpersonen gehören.
- *„Durchsicht auf Formalbeleidigungen“ statt „Zensur“.* „Zensur“ darf grundsätzlich nicht durchgeführt werden und dieser Begriff sollte auch aus dem System gestrichen werden. Vielmehr handelt es sich um eine „Durchsicht auf Formalbeleidigungen“.

4.4 Nutzungsprofile

Während des Betriebes haben wir einige Informationen über die technische Ausstattung der Nutzer aufgezeichnet. So konnten wir validieren, ob unsere anfänglichen technischen Annahmen sich bewahrheitet haben oder nicht.

Besonders haben uns die Bildschirmauflösung (Anzahl Pixel) und der verwendete Browser interessiert.

