Übungsplattform 2.0

Projektpraktikum
WS 2013/2014

Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik

Projektteam:
Jörg Baumgarten
Ralf Busch
Martin Daute
Lisa Dietrich
Christian Elze
Peter König
Florian Lücke
Felix Schmidt
Till Uhlig

Betreuer:
Dr. Werner Gabrisch

27. Februar 2014

Inhaltsverzeichnis

1 Einleitung

Das Ziel des Projekts „Übungsplattform 2.0“ war es, ein System zu entwickeln, welches zur Verwaltung, Erstellung, Abgabe und Kontrolle von Übungsaufgaben dient. Nutzer dieser Übungsplattform sind, wie bereits bei der bestehenden Übungsplattform, Mitarbeiter wie Dozenten und Kontrolleure sowie Übungsteilnehmer beziehungsweise Studenten der Martin-Luther-Universität Halle-Wittenberg. Eine Anknüpfung an das Veranstaltungsmanagement-System Stud.IP ist gegeben, dennoch ist die Plattform auch autark lauffähig. Auftraggeber war das Institut für Informatik der Martin-Luther-Universität, vertreten durch Annett Thüring, Ivo Hedtke und Sandro Wefel.

Die Übungsplattform 2.0 soll das bisherige System ersetzen. Gründe für die Entwicklung eines Nachfolgesystem sind unter anderem:

- die alte Plattform hat ihren Lebenszyklus überschritten
- die Datenbank ist nicht optimal entworfen und folglich zu langsam
- der Quellcode ist ungeeignet für die Weiterentwicklung, da an der bisherigen Übungsplattform verschiedene Gruppen arbeiteten und diese ihren jeweils eigenen Stil verwendeten
- die alte Plattform wurde ursprünglich für eine kleine Anzahl von Veranstaltungen konzipiert und ist daher nicht für das jetzige Ausmaß an Daten optimiert
- die verwendeten Konzepte und Techniken sind veraltet, die Optik ist verbesserungsfähig

1.1 Funktionen

Da wir uns nicht an dem alten System orientieren sollten, hatten wir keine Vorlagen und entwickelten die Übungsplattform von Grund auf neu. Grundlage bildeten Open-Source-Standard-Elemente unter Linux. Wir haben uns für ein Komponentensystem entschieden und für jedes Objekt (Serie, Aufgabe, Einsendung, Korrektur etc.) eine eigene Komponente erstellt. Diese kommunizieren durch eine Controllerkomponente. So können die einzelnen Elemente leicht von späteren Entwicklern ausgetauscht, entfernt, überarbeitet oder neue Elemente hinzugefügt werden. Das System läuft – wie vom Auftraggeber gewünscht – als Webservice. Ein Single Sign-On-System für Stud.IP wurde entwickelt, so dass ein direkter Zugriff ohne zusätzliches Login gewährleistet ist.

Unser System ermöglicht Dozenten das Einstellen, Ändern und Löschen von Übungsserien, Musterlösungen und Anhängen, sowie das Klassifizieren einzelner Aufgaben (zum Bei-

spiel in Bonus-, Theorie- und Praxispunkte). Ebenso sind Dozenten berechtigt einen Bearbeitungszeitraum festzulegen und sich einen Überblick über bisher eingesendete Lösungen zu verschaffen. Sie haben zusätzlich auch die gleichen Rechte wie Kontrolleure und können bei Bedarf die Übungsplattform aus Sicht eines Studenten sehen.

Kontrolleure können korrigierte Lösungen als Datei ihrer zugewiesenen Gruppen oder Aufgaben einstellen und Kommentare dazu abgeben. Musterlösungen sowie Übungsserien und Anhänge können jederzeit von einem Kontrolleur eingesehen werden. Die Lösungen bekommen den Status 'vorläufig' oder 'korrigiert'.

Studenten können die Übungsblätter ab dem Bearbeitungsbeginn einsehen und ihre dazugehörigen Lösungen einstellen, ändern und löschen und Kommentare zu den entsprechenden Lösungen abgeben. Nach Abgabeschluss ist es ihnen möglich, auf die Musterlösung zuzugreifen. Ein Student hat zudem jederzeit die Möglichkeit, seine korrigierten Lösungen einzusehen. Für ihn gibt es sowohl eine Übersicht über seine eingesendeten Lösungen als auch eine Übersicht über seine bisher erbrachte Leistung.

Neben den drei genannten Rollen (Dozent, Kontrolleur und Student) gibt es zudem Administratoren. Diese können Grunddaten einer Veranstaltung wie Veranstaltungstitel und maximale Gruppenstärke ändern und einstellen. Sie sind ebenso berechtigt, Dozenten und Kontrolleure zu verwalten. Ein oder mehrere Super-Admins legen Veranstaltungen neu an und vergeben Administratorrechte.

Unsere Übungsplattform unterstützt die Bearbeitung und Kontrolle der Übungsserien in Gruppen. Dabei lädt der Gruppenführer seine Mitglieder ein und diese müssen die Einladung akzeptieren. Aufgaben oder Gruppen können den Kontrolleuren automatisch zugewiesen werden.

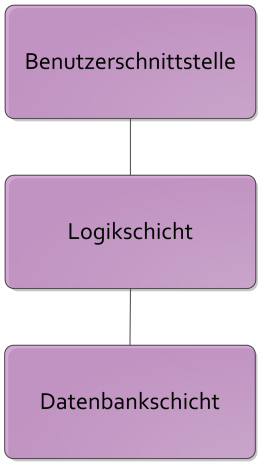
1.2 Gruppenaufteilung

Das Projekt setzt sich aus mehreren Teilgebieten zusammen, welche in den folgenden Abschnitten genauer beschrieben werden. Es handelt sich dabei um die Benutzerschnittstelle, die Logikschicht und die Datenbankschicht. Wir neun Studenten haben uns folgendermaßen auf die verschiedenen Bereiche des Projekts aufgeteilt und diese bearbeitet:

Benutzerschnittstelle:
Ralf Busch
Florian Lücke
Felix Schmidt

Logikschicht:
Martin Daute
Christian Elze
Peter König

Datenbankschicht und Dateisystem:
Jörg Baumgarten
Lisa Dietrich
Till Uhlig



2 Benutzerschnittstelle

Die Benutzeroberfläche besteht aus einer Webanwendungsschicht für Benutzer (UI) und einer Logikkomponente (GetSite), welche einen einfachen Weg darstellt, Daten für die Benutzeroberfläche in einer einzelnen Abfrage bereitzustellen.

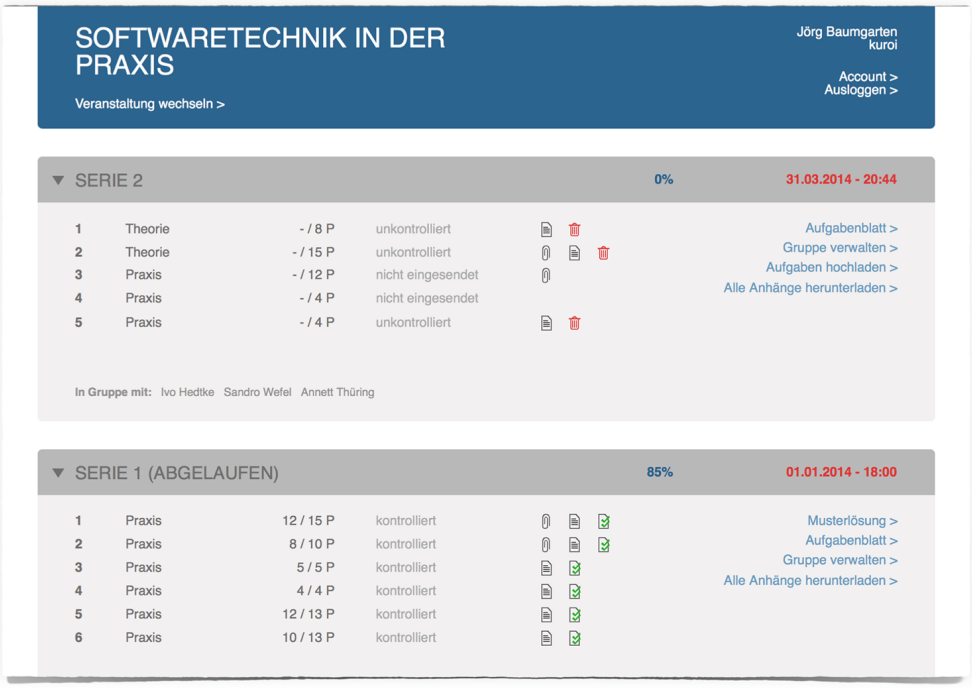


Abbildung 1: Die Übungsplattform aus der Sicht eines Studenten.

2.1 User-Interface (UI)

Die Übungsplattform sollte für die Benutzer ohne die Installation zusätzlicher Software und Erweiterungen möglich sein. Da davon auszugehen ist, dass jeder moderne Computer über einen HTML-fähigen Webbrowser verfügt, sollte eine Webanwendung entwickelt werden, welche über den Browser bedient werden kann.

Die Entscheidung für eine Programmiersprache wurde durch die Vorgabe der Markup-Sprache stark beeinflusst. Aufgrund der guten Integration fiel die Entscheidung auf PHP, auch wenn kein Gruppenmitglied tiefgehendere Kenntnisse in dieser Sprache vorweisen konnte. Der Apache HTTP Server wurde auf Grund der hohen Verbreitung

gewählt. Um die Konsistenz des Projektes zu gewährleisten, wurden diese Entscheidungen auch von den anderen Gruppen übernommen.

2.1.1 Seitenaufruf

Die Oberfläche ist aus mehreren Seiten aufgebaut, welche durch den Benutzer aufgerufen werden. Sie sorgen für eine strukturierte Darstellung der Daten aus der Datenbank und ermöglichen gleichfalls das Bearbeiten dieser Inhalte.

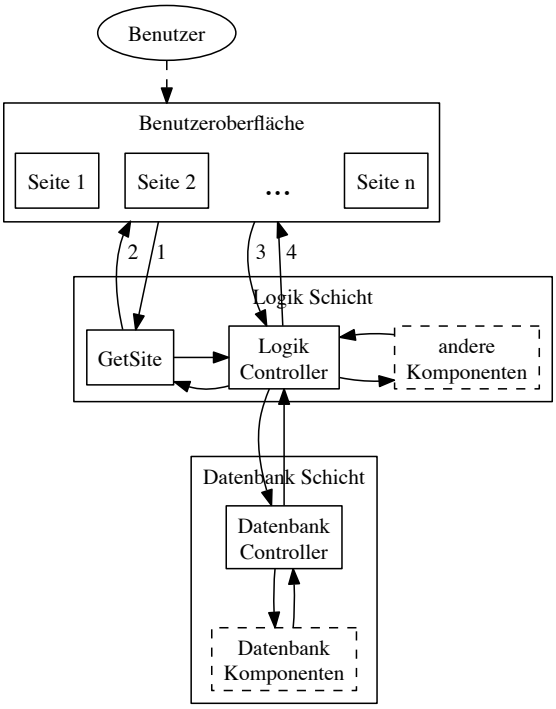


Abbildung 2: Ablauf eines Seitenaufrufes in der Benutzeroberfläche

Der Ablauf eines Seitenaufrufs ist in Abbildung 2 zu sehen. Beim Öffnen einer Seite im Browser des Benutzers wird ein Prozess des Apache Servers aktiv und führt das PHP Script für den Aufbau dieser Seite aus. Das Script fragt den Inhalt der Seite bei der GetSite Komponente ab (Schritt 1 im Bild), welche in mehreren Abfragen die benötigen

Daten über den Controller der Logik aus der Datenbank abruft. Danach gibt sie den Inhalt zurück an das aufrufende PHP Script (Schritt 2). Bei Änderungen an Daten sendet eine Seite die Anfragen an die Controller Komponente der Logik (Schritt 3), welche die Anfragen an andere Logikkomponenten und die Datenbank weiterleitet. Das Ergebnis der Anfragen wird an das PHP Script zurückgegeben (Schritt 4).

Der Grund für die Entwicklung der GetSite-Komponente wird im Abschnitt 2.2 weiter erläutert.

2.1.2 Templates

Der gewünschten Trennung von Daten und Layout entsprach auch der Wunsch der Kunden, dass System in Komponenten zu gestalten. Um diese Trennung zu verwirklichen, sollten Templates verwendet werden, welche die übergebenen Daten in HTML- und CSS-Code umwandeln, um das Layout der Seite vorzugeben. Mögliche Optionen dafür waren XML + XSLT, eine bereits existierende Template Engine oder die Verwendung einer selbst entwickelten Lösung.

Zu Beginn des Projektes wurden verschiedene Möglichkeiten verglichen, um die Daten zwischen den Schichten und Komponenten auszutauschen. Es wurden JSON und XML als Alternativen diskutiert, auf Grund der besseren Unterstützung durch PHP und des geringeren Overheads entschieden wir uns jedoch für JSON. Gleichzeitig entfiel damit auch die Möglichkeit der Verwendung von XML/XSLT zur Realisierung der Templates.

Da alle Gruppenmitglieder im Umgang mit PHP unerfahren waren, entschieden wir uns gegen die Verwendung einer fertigen Template Engine, da dies das Erlernen einer Template-Sprache mit sich gezogen hätte. Die nun verwendete Template Engine ist eine Eigenentwicklung, welche PHP verwendet und auf der Struktur der übergebenen Daten basiert. Anfangs war das System ein einfaches Suchen-und-Ersetzen-System, die Ersetzungen mussten jedoch manuell durchgeführt werden. Später automatisierten wir dieses System, um die Verarbeitung größerer Mengen von Daten zu vereinfachen. Allerdings war es nun noch immer nicht möglich, mit diesem System einfache Berechnungen durchzuführen oder Elemente zu nummerieren, weswegen entschieden wurde, dass eine weitere Überarbeitung notwendig sei. Dabei handelte es sich um die letzte größere Änderung, welche letztendlich zur aktuellen Version führte.

2.2 Logikkomponente (GetSite)

Ursprünglich sollte die Benutzeroberfläche für das Abfragen der zur Darstellung benötigten Daten zuständig sein. Es stellte sich jedoch heraus, dass dies eine sehr enge Kupplung der Datenbank mit der UI zur Folge hätte. Aus diesem Grund wurde eine Logikkomponente entworfen, welche es erlaubt, alle für die Darstellung einer Seite notwendigen Daten in einer Anfrage abzurufen. Dies verringerte die Ladezeit der Seiten erheblich, da die Anfragen der UI an die Logik so entfielen.

Anfangs wurde diese Komponente durch die Mitglieder der Logikschicht entwickelt. Da dann jedoch bei Fehlern, Verbesserungsvorschlägen oder Erweiterungswünschen unerwünschte Verzögerungen durch notwendige Absprachen entstanden sind, wurde die Verantwortung für die Komponente an die Mitglieder der UI-Schicht übertragen.

Oft stellte sich heraus, dass das Zusammenstellen der nötigen Daten sehr langsam war, was die Benutzbarkeit der Seiten einschränkte. Eine Lösung für dieses Problem bestand darin, statt mehreren kleinen Abfragen einige größere Anfragen an die Datenbank zu stellen. Ein anderer Lösungsansatz ist die Vorverarbeitung von Daten, um zum Beispiel eine bestimmte Aufgabe mit Hilfe von Indizes schneller zu finden. Eine Kombination der Ansätze lieferte in den meisten Fällen die besten Ergebnisse.

3 Logikschicht

Bei der Logikschicht handelt es sich um die Verbindungsschicht zwischen der Benutzerschnittstelle (UI) und der Datenbank (DB) bzw. dem Filesystem (FS). Dadurch ist eine leichtere Anbindung neuer Komponenten und somit neuer Funktionen möglich, welche die Kommunikation bezüglich UI und DB/FS sicherstellen, ohne dass eine dieser Schichten Informationen über die jeweils andere besitzen muss. Zudem kann das System dadurch flexibel auf mehreren Servern verteilt werden.

Mehr als zehn objektorientierte Komponenten können über eine zentrale Komponente, den Logik-Controller, angesprochen werden, welcher so eine komponenten- und auch schichtübergreifende Kommunikation und Komponentenverwaltung bereitstellt. Weiterhin trägt die Idee eines zentralen Controllers pro Schicht zur Systemsicherheit bei.

3.1 Aufbau

Die Grundlage des Logikaufbaus bildet das Komponentensystem zur leichteren Erweiterbarkeit und der Möglichkeit der Komponentenverteilung auf mehreren Servern. Darauf aufbauend ergaben sich zwei Überlegungen, nach welchem Gesichtspunkt die Komponenten erstellt werden sollen: funktionsorientiert oder objektorientiert. Da sich jedoch herausstellte, dass sich objektorientierte Komponenten leichter gruppieren ließen, haben wir uns entsprechend für diese Form entschieden.

Später stellte sich eine dritte Auswahlmöglichkeit heraus, welche aus Zeitgründen jedoch nicht mehr umgesetzt werden konnte: seitenorientierter Komponentenaufbau. Für jede in der Benutzerschnittstelle existierende Seite wäre so eine eigene Komponente entstanden, welche die jeweiligen Informationen für diese zur Verfügung gestellt hätte. Diese Idee entstand leider erst mit der Fertigstellung der speziellen GetSite-Komponente, welche vorab für Seiten alle relevanten Informationen sammelt und bereitstellt.

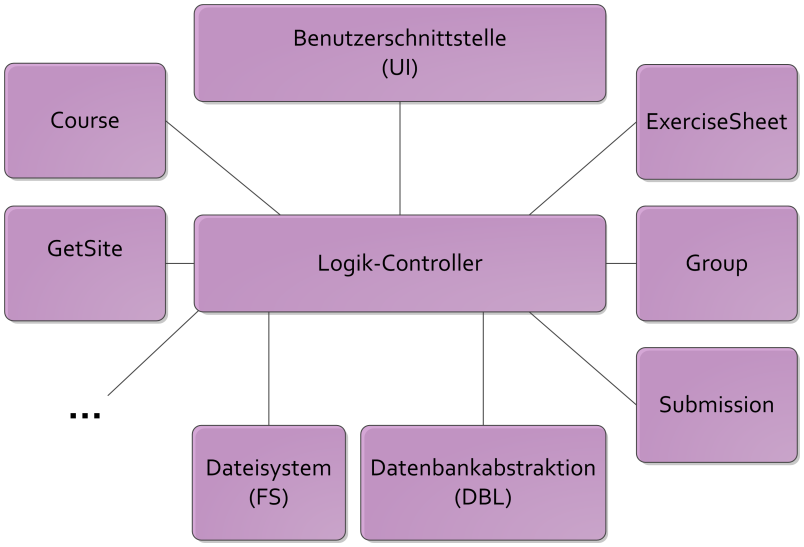


Abbildung 3: Aufbau der Logikschicht

Nachdem der komponenteninterne Aufbau festgelegt wurde, folgte die Einführung zentraler Komponenten je Schicht – die Controller – mit resultierender Verwaltungsmöglichkeit und schichtübergreifender Kommunikation als Hintergrund. Im Laufe des Projektes stellte sich dadurch zudem eine erhöhte Kommunikationssicherheit heraus: Da nur die Controller untereinander kommunizieren, kann eine vollkommen fremde Instanz nur sehr schwer Anfragen stellen oder Manipulationen durchführen.

Daraufhin wurden den Komponenten die für die Übungsplattform benötigten Funktionen zugeordnet. So entstanden auch weitere, noch nicht in Betracht gezogene Komponenten, welche sich mitunter als notwendig erwiesen.

Mit diesen Informationen konnten die ersten benötigten Parameter für die REST-Anfragen erfasst werden. Diese unterlagen oftmaligen Änderungen, da Parameter entweder fehlten oder aber überflüssig waren. Zur leichteren Komponentenverwaltung entschieden wir uns daher für eine Config-Datei pro Komponente, welche in der Datenbank erzeugt und bei Bedarf an alle Komponenten gesendet wird. Alle Komponenten erhielten so nun eine eigene Möglichkeit zur Adressierung und Verlinkung und somit auch die Option der Verteilung auf unterschiedlichen Servern.

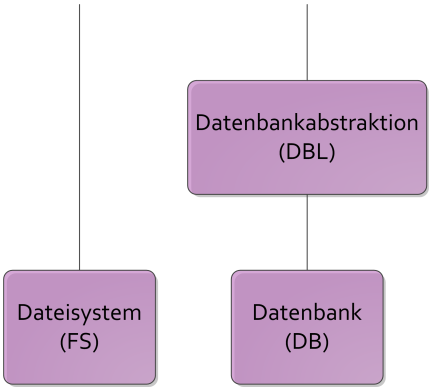
Es folgten erste Tests einzelner Anfragen an Komponenten, um diese so auf die korrekte Funktionalität zu überprüfen. Einige Fehler hierbei waren fehlende oder falsch gesetzte Parameter, welche auf veränderte Definitionen der REST-Anfragen zurückzuführen waren.

Daraufhin wurde die GetSite-Komponente konstruiert, um für die Benutzerschnittstelle und deren existierende Seiten Information bereitzustellen. Ab diesem Punkt kam – wie bereits zuvor erwähnt – erstmals der Gedanke eines seitenorientierten Komponentenaufbaus zustande, was jedoch durch den zu weiten Projektfortschritt verworfen wurde. Vorteile diese Aufbaus wären unter anderem eine übersichtlichere und auf die Seiten abgestimmte Funktionsverteilung für jede Komponente gewesen.

Als letzter großer Schritt wurden nun die Tests soweit ausgebaut, dass auf korrekte Anfragen und Antworten geprüft wurde. Hierbei traten die meisten Schwierigkeiten auf – sei es, dass falsche Parameter übergeben wurden, Funktionen falsche Werte berechneten, fehlerhafte Verlinkungen zur Datenbank bestanden oder aber auch falsche Datenstrukturen umgesetzt wurden. Der Übersicht wegen wurde daher ein Diagramm erstellt, welches die Komponenten mit ihren Funktionen und den benötigten Argumenten aufzeigt und den Stand der derzeitigen Funktionalität widerspiegelt.

4 Datenbankschicht

Dieser Teil des Gesamtsystems setzt sich aus den Teilbereichen Datenbank, der dazu gehörigen Datenbankabstraktion und dem Dateisystem zusammen.



Dabei kann jedes Teilsystem unabhängig von den übrigen betrieben und erweitert werden. Eine Verknüpfung derer entsteht durch die entsprechende Konfiguration.

Um den Zugriff anderer Schichten auf das Dateisystem und die Datenbankabstraktion zu erleichtern, wurden hier sogenannte Controller als zentrale Zugangspunkte angelegt. Diese leiten eingehende Anfragen an die verantwortlichen Komponenten in ihren Teilbereichen weiter und sollten ausschließlich genutzt werden.

4.1 Datenbank (DB)

Sowohl für die Speicherung der Daten, welche sich während des Betriebs der Übungsplattform ansammeln, als auch für die betriebsnotwendigen Daten wurde eine Datenbank für die

Nutzung mit einem MYSQL Server unter Verwendung von MYSQL Workbench 6.0 modelliert. Dabei ergab sich ein Umfang von 19 Tabellen mit über 100 Attributen.

Zur Beschleunigung von lesenden Anfragen wurden bei einigen Tabellen zusätzliche Attribute angelegt, wodurch die Notwendigkeit oft genutzter Join-Vorgänge minimiert wird. Der Inhalt dieser Spalten wird beim Einfügen einer neuen Datenzeile automatisch von der Datenbank mit Hilfe von implementierten Triggern befüllt, sodass diese Informationen nicht explizit angeben werden müssen.

Die Datenbank sollte als reiner Speicherort, ohne Wissen über den Rest des Systems, betrieben werden. Dennoch wurden aus Konsistenzgründen, zur Vereinfachung des Aufwands für andere Teilbereiche und zur Steigerung der Leistung einige Mechanismen in Form von Triggern implementiert. Es wurden beispielsweise 35 Trigger verschiedenster Art entwickelt, um sicherzustellen, dass eine Anmeldung am System nur dann möglich ist, wenn der Nutzer nicht als gesperrt vermerkt ist.

Zur Überprüfung der Langzeitstabilität des Entwurfs wurde die Datenbank während der Zeit der Entwicklung mit Beispieldaten im Umfang von rund 700.000 Datensätzen betrieben. Weiterhin wurde für eine kalkulierte Laufzeit des Systems von fünf Jahren ein lauffähiger Versuch mit über 700 Millionen Datensätzen durchgeführt.

Zudem verwaltet die Datenbank die Verknüpfung der Komponenten der Datenbankabstraktion, des Dateisystems und der Logik-Schicht. Dafür werden ausgehende Aufrufe von Komponenten als Links bezeichnet und mit einem Namen und einer Zieladresse gespeichert. Diese können dann per Aufruf über eine speziell dafür entworfene Komponente an alle im System befindlichen Komponenten verteilt werden, wodurch die zentrale Verwaltung und Konfiguration sichergestellt wird. Diese Entwurfsüberlegung ermöglicht es, das Weiterleitungsverhalten einzelner Komponenten zentral zu verändern und im Bedarfsfall zusätzliche Komponenten an entsprechenden Stellen einzuhängen, um so das System zu erweitern.

4.2 Datenbankabstraktion (DBL)

Um den Zugang zur Datenbank zu vereinfachen und den direkten Zugriff über SQL Befehle zu vermeiden, wurde eine Vereinfachung entwickelt, welche feste Befehle nach außen über eine mittels SLIM umgesetzte REST-API zur Verfügung stellt. Dazu wurden zusätzlich Datenstrukturen für die inhaltliche Darstellung und Kommunikation von Objekten entworfen, welche die DBL in Form eines JSON-konformen Objekts erwartet und in ihren Antworten ebenso auch kodiert.

Die Anfragen werden nach dem Eintreffen in SQL Befehle übersetzt und mit einer der DBL zur Verfügung gestellten Datenbankschnittstelle umgesetzt.

Dieses Vorgehen, welches direkte Datenbankzugriffe unterbindet, steigert die Austauschbarkeit sowie die einfache Erweiterbarkeit der Befehlsvielfalt. Außerdem wird an dieser Stelle sichergestellt, dass die Abfragen festgelegte Bedingungen erfüllen, wie beispielsweise die Einhaltung eines bestimmten Datentyps für Parameter des REST-Aufrufs. Darüber hinaus werden eingehende Daten von unerlaubten Inhalten bereinigt, um einen nicht ordnungsgemäßen Zugriff auf innere Systeme sowie Datenbestände zu verhindern.

Um ein umfangreiches Angebot unterschiedlichster Anfragen an die Datenbank bereitstellen zu können, wurde ein Verbund aus 20 unabhängigen Komponenten mit über 140 Befehlen entwickelt, wobei eine Komponente im Allgemeinen für die Bereitstellung von Funktionen zur Handhabung der entsprechenden Tabelle der Datenbank zuständig ist.

Zur Sicherstellung der korrekten Funktionalität der Komponenten wurden einfache, automatisierte Tests unter Verwendung des PHPUnit Frameworks entworfen, durch welche die Komponenten in ihrer Gesamtheit auf die Erfüllung von über 400 Bedingungen geprüft werden können. Es muss jedoch angemerkt werden, dass diese Tests keine ausführliche Prüfung der Funktionalität der Komponenten darstellen.

4.3 Dateisystem (FS)

Zur Speicherung von Dateiinhalten – wie beispielsweise die von Studenten eingereichten Lösungen zu Übungsaufgaben – wurden fünf weitere Komponenten entwickelt. Diese sind für die Verwaltung gespeicherter Inhalte, die Wahl des Speicherortes und der Erstellung von Archiven im ZIP-Format aus gespeicherten Inhalten zuständig.

Es war dabei ein wesentliches Anliegen, dass die redundante Speicherung von Dateiinhalten vom Dateisystem selbstständig verhindert wird. Dies geschieht durch die Benennung der gespeicherten Dateien anhand des SHA1-Hashs, welcher von der entsprechenden Datei gebildet wird.

Das Dateisystem behandelt die Dateiinhalte, ohne dabei die Herkunft oder die Bedeutung der Dateien kennen zu müssen. Zusätzlich ist es bereits in der Lage, die zu speichernden Inhalte per Konfiguration auf mehrere Speicherorte gleichmäßig zu verteilen, was durch die Entscheidung, einen Hash als Speichernamen zu verwenden, ermöglicht wurde.

5 Das Projekt aus Sicht der Softwaretechnik

5.1 Projektorganisation

Am Projekt arbeiteten insgesamt neun Studenten der Informatik. Die Koordination dieser Gruppe übernahm Felix Schmidt. Zu seinen Aufgaben gehörte es, die wöchentlichen Treffen des Teams und die (zusätzlichen) Treffen mit dem Kunden zu organisieren. Zudem stand er diesem stellvertretend für die Gruppe als direkter Ansprechpartner zur Verfügung. Bei den wöchentlichen Treffen mit dem Kunden wurden die Fortschritte des Projekts vorgestellt, Fragen gestellt sowie Absprachen mit diesem getroffen. Die Inhalte und Ergebnisse dieser Treffen wurden von Lisa Dietrich in Protokollen festgehalten.

Neben den Treffen des gesamten Teams haben sich die einzelnen Gruppen ebenfalls regelmäßig getroffen, die bis zum nächsten Teamtreffen zu erledigenden Aufgaben besprochen und zusammen Lösungen für schwerere Probleme gefunden.

Zur Organisation wurde das webbasierte Projektmanagement-Werkzeug *Trac* in Verbindung mit der Versionsverwaltung *Git* genutzt.

5.2 Erfahrungen

Anfangs gab es einige Probleme, ein Gruppenmitglied zu finden, welches die Organisation und Leitung des Projekts übernimmt, da alle Personen eher an der praktischen Arbeit interessiert waren. Letztendlich erklärte sich Felix Schmidt unter der Bedingung, trotzdem selbst Entwicklungsaufgaben ausführen zu können, bereit, die Kommunikation mit dem Kunden zu übernehmen. Der Auftraggeber zeigte sich bei Fragen und Problemen sehr hilfsbereit, so dass man sich in den wöchentlichen Treffen immer einigen konnte. Trotzdem verzichteten wir bei der Entscheidung über den Projektleiter aber auf eine gruppenübergreifende Leitung des Projektes und entschieden uns stattdessen dafür, jeden Teilbereich eigenständig zu verwalten. Dies stellte sich jedoch als problematisch heraus, da der Gesamtüberblick über das Projekt oftmals aus dem Auge verloren wurde.

In den ersten Wochen des Projekts verständigten wir uns auf Schnittstellen zwischen den einzelnen Schichten des Systems. Außerdem planten wir die Aufteilung der Komponenten, wählten PHP als Programmiersprache, beschlossen die Kommunikation zwischen den Komponenten mittels einer REST-Architektur und JSON-Objekten mittels des SLIM-Frameworks zu realisieren und entwarfen die Datenstrukturen der JSON-Objekte.

Da sich alle Gruppenmitglieder nach der Analysephase mit der Bearbeitung ihrer Aufgaben und dem Schreiben des Quelltextes beschäftigt haben, wurde die ebenso wichtige Dokumentation des Projektes teilweise vernachlässigt und musste daher während der letzten Wochen der Bearbeitungszeit zeitaufwendig nachgeholt werden.

Rückblickend betrachtet hätten viele Änderungen am Entwurf des Systems sowie an bereits fertigem Quellcode durch eine ausführlichere und bessere Planung zu Beginn des Projekts vermieden werden können. Ein besserer Entwurf war jedoch nur schwer möglich, da kein Mitglied des Projektteams auf Grund fehlender Erfahrungen genaue Vorstellungen davon hatte, auf welche Punkte besonders geachtet werden sollte und welche Komplexität die Plattform tatsächlich besaß. Weiterhin war die kurze Bearbeitungszeit ausschlaggebend für unsere Entscheidung, frühestmöglich mit der Umsetzung der Übungsplattform zu beginnen. Auf Grund des sehr engen Zeitrahmens von nur einem Semester wurde außerdem gleich zu Beginn festgelegt, dass nicht alle Punkte des Lastenheftes umgesetzt werden können und es sich bei dem Ergebnis daher noch nicht um die finale Version der neuen Übungsplattform handeln wird. Abschließend bleibt aber mit Sicherheit festzuhalten, dass das Projekt im Rahmen unserer Möglichkeiten erfolgreich beendet wurde.

Literatur

[1] Apache. http://www.apache.org/.

[2] cURL. http://curl.haxx.se/.

[3] cURL. http://curl.haxx.se/libcurl/php/.

[4] cURL. http://www.php.net/manual/de/book.curl.php.

[5] Git. http://git-scm.com/.

[6] JSON. http://www.json.org/.

[7] JSON. http://www.w3schools.com/json/.

[8] MySQL Server. http://dev.mysql.com/downloads/mysql/.

[9] PHP. http://www.php.net/manual/de/.

[10] PHP. http://www.w3schools.com/php/.

[11] PHPUnit. http://phpunit.de/.

[12] RESTful.
https://www.ibm.com/developerworks/webservices/library/ws-restful/.

[13] Slim. http://www.slimframework.com/.

[14] Trac. http://trac.edgewall.org/.

[15] MySQL Workbench 6.0. http://www.mysql.de/products/workbench/.