Design und Implementierung einer Software für online-Prüfungssysteme

Master Thesis

Author(s):
Heinrich, Peter

Publication date:
2008

Permanent link:
https://doi.org/10.3929/ethz-a-005677900

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

MASTERARBEIT

Design und Implementierung einer Software für online-Prüfungssysteme

Institut für Computational Science

Autor:
Peter HEINRICH
Legi-Nr.: 02-925-865

Leiter:
Professor Dr. Hans HINTERBERGER

Betreuer:
Markus DAHINDEN

17. August 2008

Danksagung

Für die äusserst angenehmen Arbeitsbedingungen, unter den ich diese Arbeit ausführen durfte, danke ich Herrn Professor Dr. Hans Hinterberger und seiner Gruppe. Mein Dank besonders auch an Markus Dahinden, der als betreuender Assistent erfolgreich mein andauerndes Interesse für diese Aufgabe unterstützte. Mit dem sich daraus entwickelten besonderen Arbeitsklima konnte ich mich nicht nur der Aufgabe stellen, sondern auch das Themenumfeld gründlich erarbeiten.

Abstract

This work covers the design and the implementation of a software system for online- or e-assessments. One major topic is the security of that system. It uses best practices of applied security for achieving a paper equivalent security standard. The other part describes the actual implementation in Java, as well as the construction of the server system.

Kurzbeschreibung

Diese Arbeit beinhaltet das Design und die Implementierung einer Software für online Prüfungssysteme. Besonders viel Wert hab ich dabei auf den Sicherheitsaspekt eines solchen Systems gelegt. Es wird mit anerkannten kryptographischen Methoden ein, der Papierprüfung ähnliches, Sicherheitsniveau erzeugt. Der zweite Teil der Arbeit beinhaltet die Implementierung in Java, sowie dem Aufbau einer geeigneten Server-Infrastruktur.

Inhaltsverzeichnis

1 Einleitung

Prüfungen oder Leistungskontrollen sind ein zentraler Vorgang in nahezu jeder Ausbildung. Neue, technische Errungenschaften haben breitbandig Einzug in die Ausbildung gehalten, vor allem im Bereich der multimedialen Unterrichtshilfsmittel. Prüfungen hingegen werden grösstenteils mit Stift und Papier, wie seit vielen Jahrzehnten üblich, durchgeführt. Erst ein kleiner Teil dieser Leistungskontrollen werden mit Hilfe der Informationstechnologie durchgeführt und ausgewertet. Bei Massentests wie zum Beispiel der Fahrausweisprüfung ist diese Methode bereits im Einsatz. Im akademischen Umfeld ist die Akzeptanz noch nicht sehr gross und erst allmählich sammeln Universitäten in Europa erste Erfahrungen mit diesen Systemen.

Mit dieser Masterarbeit wurde das Ziel angestrebt ein Prüfungssystem zu entwerfen, welches zunächst auf die Bedürfnisse der Gruppe von Prof. Hinterberger am Institut für Computational Science an der ETH Zürich zugeschnitten ist. Besonderer Wert sollte hierbei auf die Systemsicherheit gelegt werden, die eine integrale Rolle für den Einsatz bei benoteten Leistungskontrollen spielt.

1.0.1 Warum soll man Prüfungen am PC durchführen?

Diese durchaus berechtigte Frage habe ich mir auch am Anfang des Projektes gestellt, denn es müssen schon gewichtige Vorteile vorhanden sein, um eine gut funktionierende, anerkannte und jahrzehntelang erprobte Vorgehensweise zu ersetzen. Ein Computersystem wird in den meisten Fällen vor allem darum eingesetzt, um dem Menschen Arbeit abzunehmen bzw. diese zu beschleunigen. In Bezug auf Prüfungen geht es vor allem um die Zusammenstellung der Fragen, die Korrektur der Abgaben und nicht zuletzt auch der jahrelangen Archivierung abgelegter Prüfungen. In all diesen Bereichen kann der Einsatz von Informatikmitteln, Ressourcen schonen und Arbeit abnehmen. Ob sich auch finanzielle Aufwendungen zur Durchführung der Prüfungen reduzieren lassen, bedarf es einer Analyse, um diese Frage zu beantworten.

1.0.2 Warum sollte man Prüfungen nicht am PC durchführen?

Werden Arbeitsabläufe und Vorgehensweisen durch Software ersetzt, ergibt sich fast immer zumindest ein grosser Nachteil: Der Verlust an Transparenz. Im Fall der Leistungskontrollen ist nichts transparenter als ein von Hand beschriebenes Blatt Papier. Der gesamte Prozess der Leistungskontrolle, von der Aufgabenstellung bis zur Notenfindung, kann in jedem Punkt von Jedermann nachvollzogen werden. Das Dokument kann aufbewahrt werden und ist immer im Original verfügbar. Fälschungen, Veränderungen oder Manipulationen sind zu jedem Zeitpunkt nachweisbar und hinterlassen Spuren.
Ganz anders bei einem papierlosen Vorgehen. Je komplexer das Computersystem, desto schwerer wird es die Prozesse zu verstehen und es wird aufwendiger sich selbst oder jemand Anderen von der Sicherheit eines solchen Systems zu überzeugen.
Auch können die Auswirkungen fatal sein, sollte sich jemand des Systems unberechtigterweise bemächtigen können. Prüfungsfragen könnten vorzeitig an die Öffentlichkeit gelangen, Ergebnisse zerstört und manipuliert werden oder auch einfach nur der reibungslose Prüfungsbetrieb gestört werden.
Und genau darum geht es in dieser Arbeit. Die angesprochene unumgänglichen Probleme auf ein solches Minimum zu reduzieren, dass man ein solches System als verantwortungsvoller und sicherheitsbewusster Benutzer mit Zufriedenheit nutzen kann.

1.0.3 Sicherheitsüberlegungen

Grundsatz jeder Leistungskontrolle muss auch Fairness darstellen. Alle sollten unter den gleichen Bedingungen geprüft werden und Manipulationen sollten von allen Seiten her ausgeschlossen sein. Der Sicherheitsanspruch an ein solches System muss ja nicht unbedingt grösser sein als der einer Papierprüfung aber auch keinesfalls geringer. So möchte ich zum Beispiel Prüfungsabgaben mit Dokumentencharakter, welcher auch als solcher vom Gesetzesgeber akzeptiert wird. Manipulationen oder Fehler, die eine Prüfungsabgabe verändern, sollen erkannt und belegt werden können.

2 Anforderungen

Am Institut für Computational Science werden zum jetzigen Zeitpunkt jährlich über eintausend Prüfungen [4] alleine im Bereich Informatik-Grundlagen durchgeführt. Dabei handelt es sich zurzeit um reine Multiple-Choice-Prüfungen die auf Papier durchgeführt werden. Diese Prüfungen sollen nun am PC durchgeführt werden. Da kein, sich am Markt befindliches, Softwaresystem alle Anforderungen zufrieden stellend erfüllen kann, hat man sich hier zu einer Neuentwicklung entschieden. Aus Sicht des Departements werden zwei Systeme benötigt: Zum einen möchte man ein System an welchem man Prüfungen am PC papierlos durchführen kann und die Korrektur automatisch erfolgt, zum anderen möchte man ein System zum kollaborativen Erstellen neuer Prüfungsfragen. Kollaborativ soll hierbei heissen, dass eine Aufgabe iterativ von mehreren Personen erstellt und verbessert werden kann. Dazu sollen alle Aufgaben Revisionsnummern erhalten. Da der gesamte Prozess der Aufgabenerstellung transparent bleiben soll, möchte man auch später noch auf die alten Revisionen Zugriff haben. Kommentare sollen helfen, diesen Prozess zu erleichtern und zu dokumentieren. Vorerst soll das System nur am Institut selbst Einsatz finden, später sollen aber auch andere Institute oder Departemente von anderen Hochschulen das System nutzen können. Die nun folgende Anforderungsanalyse ist durch viele Gespräche mit den Verantwortlichen für diese Prüfungen entstanden vor allem aber durch die Informationen von Herrn Markus Dahinden.

2.1 Verwaltungssystem

Das Verwaltungssystem soll ausschliesslich Dozierenden, Assistenten/innen und Hilfsassistenten/innen zur Verfügung stehen. Das folgende Use-Case Diagramm zeigt eine Übersicht der wichtigsten Funktionalitäten:

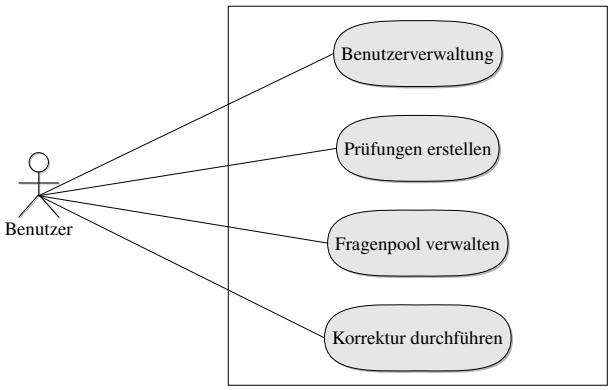


Abbildung 2.1: Grundlegende Funktionalität des Verwaltungssystems

2.1.1 Fragenpool

Die Verwaltung des Fragenpools stellt dabei den zentralen und wichtigsten Use-Case dar. Die Aufgaben sollen dabei an einem zentralen Ort gespeichert sein und archiviert werden. Mit Hilfe des nächsten Use-Case Diagramms sollen die dazu nötigen Funktionen näher untersucht werden:

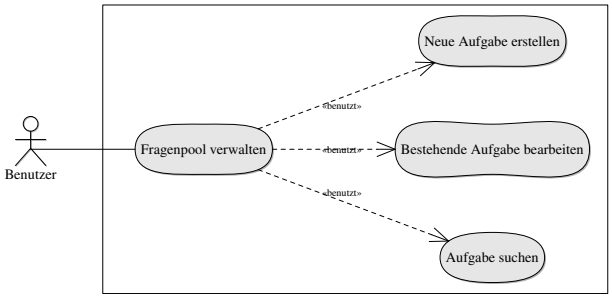


Abbildung 2.2: Funktionen der Fragenpool-Verwaltung

Neue Aufgaben erstellen

Um eine neue Aufgabe zu erstellen sind mehrere Schritte notwendig. Die Frage muss formuliert werden und die dazu nötigen Grafiken eingebunden werden. Der Benutzter soll dies mit einer grafischen Schnittstelle vornehmen können und soll dabei stets eine Vorschau der Aufgabe, wie sie in der Aufgabe Einsatz findet, zur Verfügung haben. Bilder oder Grafiken werden vom Benutzer in einem geeigneten Format (jpeg,gif oder png) zur Verfügung gestellt. Um die Aufgabe zu komplettieren müssen mindestens 5 Antworten erstellt werden, die entweder richtig oder falsch sind. Die Antworten werden dazu in ähnlicher Weise wie die Fragen erstellt. Ebenfalls können auch die Antworten Bilder enthalten.

Aufgaben bearbeiten

Um eine Aufgabe zu bearbeiten soll der Benutzer das gleiche Interface wie zum Erstellen der Aufgaben zur Verfügung haben. Einmal gespeicherte Aufgaben dürfen aber nicht einfach abgeändert werden. Es soll hierzu eine Kopie der Aufgabe mit neuer Revisionsnummer erzeugt werden. Damit erhält man zum einen die Nachvollziehbarkeit der Änderungen und zum anderen einen Schutz davor, dass eine Frage, welche schon geprüft wurde, nachträglich verändert wird. Werden Änderungen gespeichert, also eine neue Revision angelegt, so soll dies mit einem Kommentar begründet werden, wie man es auch von anderen Versionierungssystemen her kennt wie zum Beispiel CVS [1]. Es soll möglich sein, alle Revisionen einer Aufgabe einzusehen sowie alle zugehörigen Kommentare. Grafiken können aus gleichem Grund nicht gelöscht oder verändert werden. Soll eine überarbeite Grafik in einer neuen Revision verwendet werden, so wird einfach die neue Grafik in die Aufgabe eingebunden, die alte Grafik bleibt unverändert im System gespeichert, um auch bei alten Versionen Konsistenz zu erreichen.

Aufgaben suchen

Das System soll für verschiedenste Zwecke eine Suchmöglichkeit für bereits erfasste Aufgaben anbieten. Man soll nach mehreren Parametern der Aufgabe suchen können. Dazu zählt neben dem Namen der Aufgabe auch ihre Schwierigkeit, Qualität oder der Name des Erstellers. Die Aufzählung der Suchkriterien hat zu diesem Zeitpunkt nicht den Anspruch auf Vollständigkeit.

[1] `http://www.nongnu.org/cvs/` Concurrent Versions System

2.1.2 Prüfungen

Das System soll eine Möglichkeit bieten, Prüfungen aus den erfassten Fragen zu erstellen. Dazu soll der verantwortliche Assistent mit Hilfe der beschriebenen Suchmöglichkeit Aufgaben aus dem Pool auswählen können, um somit eine Prüfung zusammenzustellen. Pro gewählter Aufgabe kann er dabei aus den vorhandenen Antworten genau 5 auswählen, um so eine prüfungsfähige Aufgabe festzulegen. Des Weiteren muss die Prüfungsdauer sowie das Zeitfenster, in welchem die Prüfung abgelegt werden kann, angegeben werden. Nur innerhalb dieses Zeitfensters könne die Prüflinge später die Prüfung ablegen bzw. auf die Prüfung zugreifen. Der Zeitraum kann sich je nach Art der Prüfung von einigen Stunden (zählende überwachte Prüfung) bis mehrere Tage oder Wochen (Probeprüfung) erstrecken. Um einen sicheren Prüfungsbetrieb zu ermöglichen, muss einer Prüfung eine Liste von zugelassenen Studenten hinzugefügt werden. Diese soll manuell eingegeben werden können oder als CSV [2] im Block ladbar sein. Die Liste soll bis zum Beginn des Prüfungsfensters editierbar sein. Mit Beginn des Prüfungsfensters dürfen generell keinerlei Änderungen an der Prüfung mehr möglich sein.

2.1.3 Korrektur

Um eine Bewertung zu ermöglichen bedarf es einer Korrektur der abgelegten Prüfungen. Diese kann und soll automatisch erfolgen. In der ersten Ausbaustufe soll es lediglich möglich sein die Ergebnisse der erbrachten Leistungen in eine CSV Datei zu exportieren. Später soll dann auch noch eine teilautomatisierte Korrektur durchführbar sein, in der zum Beispiel einzelne Aufgaben im Nachhinein aus der Bewertung ausgeschlossen werden können.

2.1.4 Benutzerverwaltung

Da das System schützenswerte Informationen enthält, sind Zuganskontrollen unumgänglich. Daher soll das System über eine rollenbasierte Benutzerverwaltung verfügen. Ein Benutzer, der sich am System anmelden darf, ist zwingend einer oder mehreren Rollen zugeordnet. Folgende Rollen sind vorgesehen:

- Administrator: Der Administrator verwaltet die Benutzerkonten der Professoren. Darüber hinaus wartet er das System.
- Professoren: Die Professoren verwalten die Benutzerkonten der Assistenten. Sonst haben sie alle Recht der Assistenten.

[2] Comma separated Values - Ein für den Austausch strukturierter Daten übliches Dateiformat

- Assistenten: Die Assistenten verwalten die Benutzerkonten der Hilfsassistenten. Sie dürfen den Fragenpool bearbeiten sowie Prüfungen erstellen und korrigieren.
- Hilfsassistenten: Die Hilfsassistenten dürfen nur Aufgaben erstellen und ihre eigenen bearbeiten. Sonst steht ihnen keinerlei Funktionalität zu.

2.2 Prüfungssystem

Das Prüfungssystem beinhaltet lediglich einen zentralen Use-Case: Das Lösen einer Prüfung.

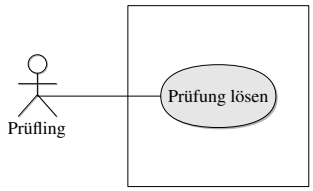


Abbildung 2.3: Grundlegende Funktionalität des Prüfungssystems

Dieser Use-Case kann jetzt weiter aufgefächert werden, um das Vorgehen zu erklären. Grundsätzlich soll der Ablauf einer solchen Prüfung sich nicht stark von einer papierbasierten Version unterscheiden. Ein Prüfungsablauf gliedert sich normalerweise wie folgt:

- Kontrolle der Authentizität des Studenten durch die Prüfungsaufsicht.
- Zur Kenntnisnahme des auf dem Deckblatt angegebenen Prüfungsablaufs.
- Lösen der einzelnen Aufgaben.
- Unterschreiben der Prüfung unter Angabe von Namen und Legi-Nummer.
- Abgeben der Prüfung.

Die Kontrolle der Authentizität wird man wohl kaum dem Computer überlassen können und wollen, schon deshalb muss auch bei einer online Prüfung eine Prüfungsaufsicht anwesend sein, um den Prozess zu überwachen. Alle weiteren Schritte soll der Prüfling dann am PC durchführen können. Das erweiterte Use-Case Diagramm sieht folgendes vor:

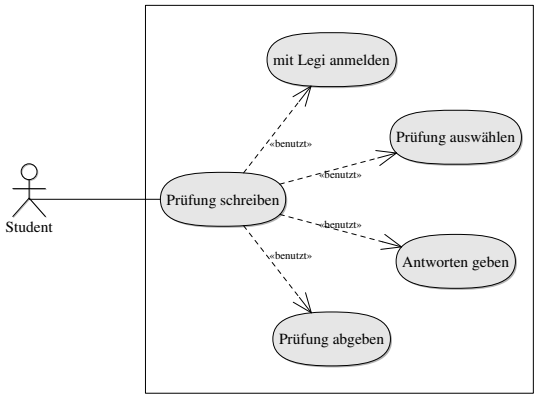


Abbildung 2.4: Funktionalität des Prüfungssystems

Mit Legi authentifizieren

Da die Prüfung nach Abgabe dem Student zugeordnet werden muss ist es erforderlich, dass mindestens die Legi-Nummer dem System bekannt gegeben wird. Die eingegebene Legi-Nummer soll während der Prüfung immer am Bildschirm sichtbar sein so, dass sie von der Aufsicht kontrolliert werden kann.

Prüfung auswählen

Aufgrund der eingegebenen Legi-Nummer prüft das System zu welchen Prüfungen der Kandidat zugelassen ist. Sollte mehr als eine Prüfung zu diesem Zeitpunkt möglich sein, soll der Kandidat eine Prüfung wählen können. Anschliessend wird dem Prüfling das Deckblatt der Prüfung angezeigt, welches den Prüfungsablauf beschreibt und wichtige Informationen wie zum Beispiel die Prüfungsdauer enthält. Mit dem Akzeptieren dieser Bedingungen startet die Uhr und dem Prüfling werden die Aufgaben präsentiert.

Antworten geben

Der Kandidat soll die Aufgaben wie bei einer Papierprüfung in beliebiger Reihenfolge lösen können, er kann sich bei Bedarf mehrfach bei einer Antwort umentscheiden.

Prüfung abgeben

Entweder auf Wunsch oder nach Ablauf der Zeit soll der Kandidat die Prüfung abgeben. Dazu soll er alle Antworten noch einmal durchsehen können (ohne

weitere Änderungsmöglichkeit) und anschliessend die Prüfung als komplettes Dokument abgeben.

Allgemeine Anforderungen an das Prüfungssystem

Das Prüfungssystem soll einen möglichst ungestörten und der Papierprüfung vergleichbaren Prüfungsablauf garantieren. Dazu gehört auch, dass eine Prüfung bei Ausfall eines Teils der EDV-Anlage möglichst uneingeschränkt fortgeführt werden kann. Für den Extremfall soll dazu immer ein Satz Prüfungen als Papierversion vorhanden sein, um notfalls darauf zurückgreifen zu können. Zum Beispiel bei einem Stromausfall. Jede am System abgegebene Antwort soll sofort gespeichert werden, um im Fehlerfalle zumindest eine Teilabgabe zu erhalten. In solch einem Fall erfolgen alle weiteren Abgaben danach auf Papier. Im Falle eines kleineren Defektes, der nur einzelne Arbeitsstationen betrifft, soll die Prüfung an jedem anderen zur Verfügung stehenden Arbeitsplatz fortsetzbar sein. Die bisher gegebenen Antworten sollen auch am neuen Arbeitsplatz sichtbar und wieder änderbar sein. Der Sicherheitsaspekt, der mit der elektronischen Übertragung und Speicherung dieser Daten verbunden ist, wird explizit im Kapitel „Sicherheitsanalyse“ abgehandelt.

3 Markt

Zwar hat diese Arbeit klar das Ziel der Neuentwicklung eines online-Prüfungssystems aber dennoch sollte man einen kurzen Blick auf den Wettbewerb werfen, alleine schon um diese Arbeit zu rechtfertigen.
Die von mir betrachteten Systeme sind „Questionmark Perception“, „Moodle“, „Olat“ und „Illias“, wobei die Firma Questionmark scheinbar der Marktführer ist und in meiner Auswahl der einzige kommerzielle Anbieter. Die anderen Projekte sind allesamt Open-Source und somit frei verfügbar. Alle Projekte haben eines gemeinsam: Sie bieten viel mehr Funktionen als die von uns benötigten vor allem im Prüfungsbereich. So können alle Produkte eine Vielzahl von möglichen Fragentypen anbieten und auswerten. Die Liste reicht von einfachen Multiple-Chioce-Antworten bis hin zu Lückentexten. Als Client, um eine Prüfung durchzuführen, verwenden alle Systeme ausnahmslos einen Web-Browser. Der Einsatz eines so genannten Lock-Down-Borowser [5] wird meistens empfohlen, Questionmark stellt laut ihrer Webseite [1] sogar selbst einen Browser mit gesteigerter Sicherheit zur Verfügung. Perception ist das am meisten spezialisierte Tool aus der Reihe. Es konzentriert sich sehr stark auf die Erstellung, Verteilung und Durchführung von Tests. Die anderen Systeme haben Ihren Schwerpunkt eher im e-learning-Bereich und damit auch eher in Richtung Kursverwaltung und Administration. Keine Software konnte gefunden werden, welche die geforderte Versionierung der Aufgaben in einem zufriedenstellenden Masse bereitstellt.
Was ich sehr schade und bemängelnswert finde ist die Tatsache, dass der gesamte Sicherheitsaspekt sehr oberflächlich und oftmals fast populär präsentiert und diskutiert wird. Meistens gibt man sich mit einer SSL-geschützten Verbindung [11] zufrieden und man brennt nach der Prüfung eine CD mit den Logfiles und Antworten. Ich konnte gerade diese Aspekte mit einem Vertreter eines führenden Herstellers solcher Software aus Deutschland [2] erörtern.
Dabei konnte ich zu der Erkenntnis gelangen, dass man sich zwar dem Sicherheitsaspekt bewusst ist aber mangels Präzedenzfälle und Klagen mit einem relativ geringen Sicherheitsmass zufrieden gibt. So begnügt man sich zum

[1] `http://www.questionmark.com/deu/perception/index.aspx` Aug. 2008
[2] Gespräch am 18.6.2008

Beispiel mit einer Checksumme über den Daten, um deren Unversehrtheit zu verifizieren, welche man zusammen mit diesen Daten in der Datenbank aufbewahrt.

4 Sicherheit

Um eine ausreichende Sicherheit bei einem solchen System herzustellen, bedarf es nicht einer einzelnen Massnahme wie zum Beispiel SSL geschützte Verbindungen, sondern einem gesamten Sicherheitskonzept. Mit dem Begriff Sicherheit in diesem Szenario sollen Datenschutz, Vertraulichkeit, Datenintegrität und Verfügbarkeit gemeint sein. Vor allem die Vertraulichkeit und die Datenintegrität stellen hierbei die beiden zentralen Punkte dar.

4.1 Das zu schützende System

Ohne die Spezifikation vorwegzunehmen soll kurz dargestellt werden, wie der grundsätzliche Systemaufbau aussehen soll. Die Architektur wir dabei im Wesentlichen identisch zu fast allen sich auf dem Mark befindenden Prüfungssystemen sein. Folgende Komponenten werden mit Sicherheit vorhanden sein:

Prüfungs-PCs

Es wird in jedem Fall N PCs geben, auf denen gleichzeitig Prüfungen abgenommen werden. Dabei kann davon ausgegangen werden, dass es sich hierbei um handelsübliche Hardware im Desktopbereich handelt ohne besonderen Ausfallschutz oder redundanter Komponenten. Pro PC wird je Durchlauf genau eine Prüfung abgenommen.

Admin-PCs

Es gibt einen oder mehrere PCs von denen aus das System administriert wird. Dazu zählt neben der Systemadministration auch das Erstellen von Frage und Prüfungen.

Server

Um die Daten zentral speichern und archivieren zu können bedarf es einem Server-System. Es kann, muss aber nicht, aus mehreren Einzelsystemen bestehen.

Netzwerk

Es wird eine Netzwerk-Infrastruktur vorausgesetzt, die die Systeme miteinander verbindet. Es ist davon auszugehen, dass das Mithören des Netzwerkverkehrs mit einfachsten Mitteln möglich ist.

4.2 Physikalische-Sicherheit

Grundlage jedes Software-Sicherheitskonzepts ist die physikalische Sicherheit der darunterliegenden Hardware. Ist der Server öffentlich zugänglich, sind zum Beispiel alle softwareseitigen Bemühungen um Verfügbarkeit und Datenschutz nahezu nutzlos. Daher sollen hier kurz für die Systemkomponenten Vorschläge bezüglich der physikalischen Sicherheit dargestellt werden.

4.2.1 Prüfungs-PCs

Die Prüfungs-PCs sollten in einem Raum untergebracht werden, in dem Prüfungen ohne äussere Störeinflüsse abgehalten werden können (Lärm, Beleuchtung, Klima, etc.). Die PCs sollten mit Vorhängeschlössern an den Gehäusen gegen Manipulation oder Diebstahl gesichert werden.

4.2.2 Admin-PCs

Die Admin-PCs sollten sich grundsätzlich nur in verschliessbaren Büroräumen befinden. Hier wird jeder Assistent oder Benutzer im Allgemeinen zur Erbringung der physikalischen Sicherheit selbst angehalten.

4.2.3 Server

Der Server hat aufgrund seiner enthaltenen Daten einen hohen Schutzbedarf. Dieser muss sich daher in einer zugangskontrollierten Zone (zum Beispiel in einem speraten Serverraum) befinden. Vorzugsweise ist er in diesem Raum nochmals in einem abschliessbaren Schrank untergebracht. Der Raum muss den Server auch vor allen erdenklichen Umwelteinflüssen wie zum Beispiel Wassereinbruch oder Blitzschlag schützen. Um einen zuverlässigen Betrieb zu gewährleisten, muss der Server Notstrom versorgt sein und in einer klimatisierten Umgebung von nicht über 26°C [6] betrieben werden.
Wichtige und Ausfall gefährdete Komponenten sind nach Möglichkeit redundant auszuführen, zumindest die Festplatten, besser noch zusätzlich das Netzteil.

4.2.4 Netzwerk

Das Netzwerk wird in den meisten Fällen gegeben sein, Änderungen daran werden kaum möglich sein. Jedoch sollte unter allen Umständen ein drahtbasiertes Netzwerk zum Einsatz kommen, damit ein Angreifer nur mit Manipulation der Infrastruktur den Datenverkehr manipulieren kann. Am Prüfungsnetzwerk sollten darüber hinaus keine privaten bzw. unkontrollierten PC-Systeme angeschlossen werden.

4.3 Organisatorische Sicherheit

Unter organisatorischer Sicherheit versteht man, dass das Betriebspersonal auf die Sicherheitsmassnahmen geschult und sensibilisiert wird diese auch einzuhalten.

4.3.1 Allgemeine Massnahmen

- Verantwortliche und deren Aufgaben sind festzulegen.
- Wartung und Reparaturarbeiten müssen protokolliert werden. Gegebenenfalls ist die Dokumentation anzupassen oder zu erweitern.
- Schlüssel oder Badges müssen verwaltet und deren Herausgabe geregelt sein.
- Es sind Stellvertreter zu benennen, die Aufgaben im Falle von Abwesenheit übernehmen können (z.B. Hardware-Wartung).
- Datenträger aller Art, die Daten von diesem System enthalten könnten, sind nach Gebrauch entweder geschützt zu verwahren oder müssen physikalisch zerstört werden.

4.3.2 Umgang mit Passwörtern

- Verwendete Passwörter müssen mindestens 6 Zeichen lang sein und eine genügend grosse Komplexität aufweisen.
- Passwörter dürfen niemals aufgeschrieben werden.
- Durch Passwörter freigeschaltete Systeme dürfen nicht unbeaufsichtigt bleiben. Das automatische Sperren des Bildschirmes kann hier Abhilfe schaffen.

4.3.3 Routinekontrolle vor Prüfungen

- Bei Beginn der Prüfung muss sichergestellt werden, dass alle eingesetzten Sicherheitsmassnahmen unversehrt sind. Alle Wechseldatenträger müssen von den Prüfungssystemen entfernt werden.

4.4 Software Sicherheit

Wie schon in vorherigen Kapiteln erwähnt, ist eine der Papierprüfung vergleichbare Sicherheit nötig, damit eine Prüfung rekursfähig ist und damit überhaupt erst für eine benotete Leistungskontrolle an der ETH eingesetzt werden kann. Was für Sicherheitsziele sollen erreicht werden? Ich erhebe nicht den Anspruch, dass es keine Schwachstellen irgend einer Art geben darf, denn das wäre ein utopisches Unterfangen, wie bei jedem etwas komplexeren Softwaresystems. Ich kann aber fordern, dass eine Verletzung der Integrität der gespeicherten Daten auf jeden Fall bemerkt oder nachgewiesen werden kann. Das ist auch äusserst wichtig bei einem System, mit dessen Hilfe unter Umständen über die Zukunft von Prüflingen entschieden wird.
Folgende Sicherheitsziele können formuliert werden:

- Der Student soll im Zweifelsfalle beweisen können, dass die benotete Leistung nicht der abgegebenen Version entspricht.
- Der Assistent soll die Authentizität der Antworten ebenfalls verifizieren können, um Systemfehler oder Manipulationen auszuschliessen.
- Nicht autorisierte Personen dürfen keinen Zugriff auf den Fragenpool erlangen.
- Ein Angreifer darf nicht unbemerkt Übertragungen oder Datenbestände manipulieren können.

4.4.1 Begriffserklärung

Papiersicherheit

Wie der Begriff schon vermuten lässt, soll ein elektronisches Dokument im Umgang mindestens so sicher sein wie ein handgeschriebenes und unterschriebenes Papierdokument. Dabei steht vor allem die Fälschungssicherheit und damit die Echtheit im Vordergrund. Das Dokumentenformat der Wahl ist hier das PDF. Erstens kann es fast auf jeder Architektur angezeigt und gedruckt werden

und zweitens ist das Format auch von dem Gesetzgeber als Format für Dokumente anerkannt. Zumindest wenn es durch eine gültige Signatur geschützt wird, doch dazu später mehr. Am Ende einer Prüfung sollen also alle abgegebenen Antworten zu den Fragen inklusive der Aufgabenstellungen zusammenhängend in einem PDF vorliegen. Dieses repräsentiert dann die abgegebene Prüfung.

Digitale Unterschrift

Digitale Signaturen benutzen meist asymmetrische Kryptographie, wie in [10] beschrieben. Dabei wird über den zu signierenden Text ein Hashwert gebildet. Es muss sich dabei natürlich um ein Ein-Weg-Hashverfahren handeln. Ein solcher möglicher Algorithmus ist zum Beispiel MD5. Dieser Hash wird dann mit dem Private-Key der unterzeichnenden Person verschlüsselt. Das Resultat wird einfach zusammen mit dem Dokument aufbewahrt. Die digitale Unterschrift ist nicht direkt in das Dokument eingebettet, im Gegensatz zu echten Unterschriften. Das muss aber auch so sein, da man bei einem elektronischen Dokument Original und Kopie ohnehin nicht unterscheiden könnte.
Man geht davon aus, dass es nicht möglich (extrem schwer) ist ein schon bestehendes und unterschriebenes Dokument sinngebend so zu verändern, so dass sich der gleiche Hashwert errechnet. [7]
Um die Unterschrift und die Unversehrtheit des Dokumentes zu verifizieren erzeugt man in gleicher Weise wieder einen Hash von dem Dokument. Mit dem allgemein bekannten Public Key des Probanden kann die mitgelieferte Signatur entschlüsselt werden. Ist der entschlüsselte Hash identisch mir dem selbst berechneten, so ist das Dokument authentisch und unverändert.

Sichere Kanäle

Einen Kanal bezeichnet man als sicher, wenn die Gesprächspartner zum einen authentisch sind und zum anderen die Kommunikation geheim bleibt. Genau das wollen wir für den gesamten Netzwerkverkehr. Geheimhaltung mit starker Verschlüsselung zu erzeugen ist nicht weiter schwierig. So unterstützt praktisch jedes System (zum Beispiel auch ein Web Client) die Möglichkeit, Verbindungen durch SSL zu schützen. Um auch Authentizität herzustellen muss sowohl der Server den Client identifizieren können als auch umgekehrt.

4.4.2 Public Key Infrastruktur

Um diese digitalen Unterschriften zu ermöglichen muss zunächst einmal eine PKI (Public Key Infrastruktur) aufgebaut werden. Normalerweise erzeugt man sich ein Private/Public Schlüsselpaar und lässt dann den Public-Key von einem akkreditierten Zertifikationsunternehmen (z.B. Thawte, Verisign ...) beglaubigen. Das hätte den Vorteil, dass man mit diesem beglaubigten Key auch gleich alle anderen Unterschriften tätigen könnte zum Beispiel seine E-Mails signieren. Allerdings ist dieses Prozedere mit Investitionen verbunden und liegt im Bereich von 100-200 CHF [1] pro Jahr und Zertifikat. Die andere Möglichkeit ist, dass man so genannte self-signed Zertifikate benutzt.
Dazu erzeugt man sich ein Key-paar (CA.key / CA.cert). CA steht hierbei für Certificate Authority. CA.cert ist öffentlich und kann von jedermann zur Kontrolle der Zertifikate benutzt werden. CA.key ist der private Schlüsselteil und geniesst damit den allerhöchsten Schutzbedarf. Dieser Key sollte aus Sicherheitsgründen offline (z.B. als USB-Stick) verwahrt werden. Mit Ca.key können dann bei Bedarf andere Schlüsselpaare beglaubigt werden.

Erstellung eines Schlüsselpaares für einen Prüfling

Ein Sicherheitsexperte würde jetzt sicherlich vorschlagen, dass jeder Prüfling sein Schlüsselpaar selbst generiert und sich seinen Public-Key dann beglaubigen lässt. Nur so könnte der Prüfling sicher sein, dass sein private-Key nicht in falsche Hände gerät. Praktisch ist das so natürlich nicht durchführbar. Möglich ist hier, dass es eine „Black-Box“ gibt, in welche der Student seine Legi und einen USB-Stick steckt, auf welchem dann anschliessend das beglaubigte Schlüsselpaar gespeichert ist. Der generierte Public-Key verbleibt in der Black-Box und wird später zur Verifikation der unterschriebenen Dokumente benutzt.

Neben dem besagten Schlüsselpaar wird natürlich auch beglaubigt, dass der erzeugte Key zu der von der Legi gelesenen Legi-Nr. in Relation steht. Die Kontrolle des Fotos auf der Legi und damit die Authentizität des Studenten muss in jedem Fall durch einen Menschen verifiziert werden, sonst wäre das System in dieser Form nutzlos.
Diese Black-Box muss natürlich vertrauenswürdig sein, spezielle Hardware wäre auch hier wünschenswert. In der Praxis wird aber auch ein kleiner, separater Laptop reichen, der sonst für nichts anderes benutzt wird und auch nicht vernetzt ist. Es muss sich also um ein Inselsystem handeln.

[1] `https://www.thawte.com/ssl-digital-certificates/ssl123/index.html`

Damit ist jetzt ein Stand erreicht, an dem jeder Prüfling Dokumente unterschreiben kann und das System kann diese Unterschrift zweifelsfrei überprüfen und noch wichtiger auch die Unveränderheit des Dokumentes seit der Unterschrift.
Folgende Grafik soll den Vorgang illustrieren:

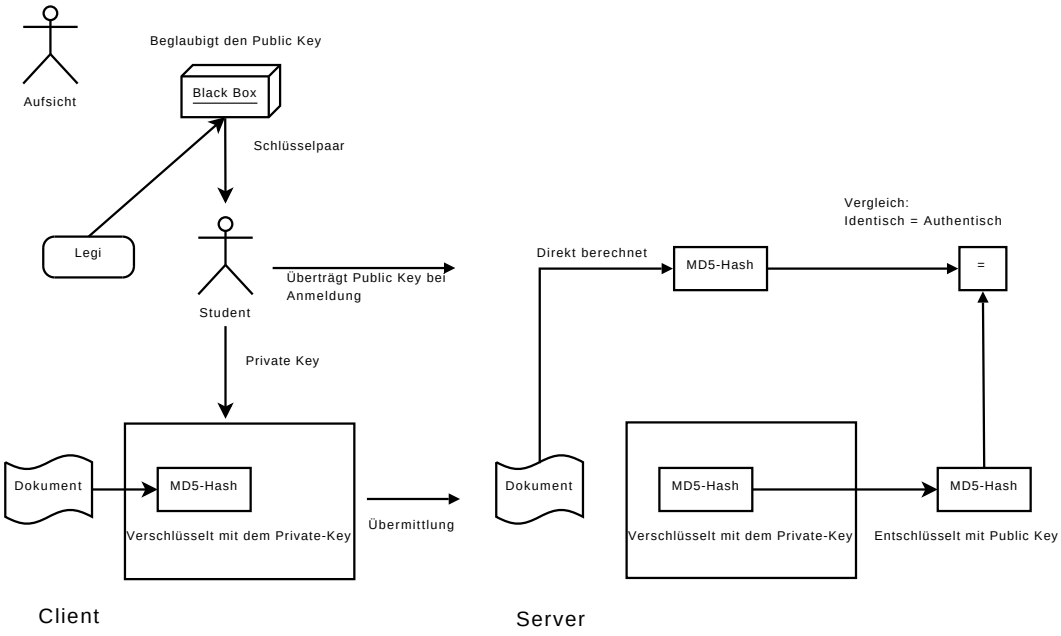


Abbildung 4.1: Übersicht Zertifikate und Signaturen

Identitätserkennung des Server

Der Server verfügt hierzu über ein beglaubigtes SSL-Zertifikat. Somit kann der Client die Authentizität des Server bei der Initiierung der Verbindung sicherstellen.

Identitätserkennung des Prüflings

Aufgrund der Tatsache, dass alle vom Client gesendeten Objekte signiert sind ist es nicht erforderlich den Client als solchen zu identifizieren, weil ja aufgrund der Signatur jederzeit der Benutzer des Systems erkennbar ist.

PDF-Dokumente und automatische Korrektur

Ein Problem, das jetzt noch bleibt, ist die Tatsache, dass sich so ein PDF relativ schlecht automatisch korrigieren lässt. Denn je nach Erzeugungsverfahren ist es schlicht ein Bild, das die Antworten enthält. Das PDF ist daher eher für eine manuelle Kontrolle und für die Archivierung geeignet. Des Weiteren soll ja das PDF auch erst am Ende einer Prüfung gesamtheitlich erzeugt werden, man möchte aber jede Antwort sofort zentral speichern, um bei möglichem technischen Versagen der Anlage wenigstens die Zwischenergebnisse vorliegen zu haben. Also ist es naheliegend auch für die Zwischenabgaben je ein signiertes Dokument zu erzeugen. Aus den genannten Gründen kann das kein PDF sein, es genügt aber einfach eine Datenstruktur mit solch einer Signatur zu versehen, bevor diese an den Server geschickt wird. Eine Signatur über das Tupel (Legi-Nr., Aufgaben-Text, Antwort-Text, Antwort, Datum, Sequenznummer) genügt bereits zu diesem Zweck. Somit ist auch nicht nur die Antwort signiert, sondern auch die Aufgabenstellung. Damit wird überwacht, dass nicht durch einen technischen Fehler die Beziehungen zwischen Antworten und Aufgaben vertauscht wurden. Das Datum ist wichtig, da man sich ja bei der Antwort mehrmals umentscheiden darf. Die zuletzt eingegangene Antwort je Aufgabe ist dann Grundlage für die Korrektur. Die Sequenznummer ist eine bei 0 beginnende Aufzählung der abgesendeten Antworten. Finden sich später Lücken in den Sequenznummern, so sind Antworten verloren gegangen.

4.4.3 Vertrauenszonen

Die Trustzone, also die Zone, in der man allen beteiligten Instanzen vertrauen muss, sollte so gering wie möglich sein. Mit den zuvor genannten Methoden kann die Trustzone auf die Client-PCs und die „Black-Box“ zur Schlüsselgenerierung beschränkt bleiben. Das Netzwerk entfällt aufgrund der sicheren Kanäle. Der Server im Prinzip auch, wenn es nur um die Durchführung der Prüfung geht. Da der Server aber wegen seiner gespeicherten Daten, vor allem dem Fragenpool, sicherheitsrelevant ist, ist er sehrwohl innerhalb der Trustzone, zumindest aus Sicht des Lehrkörpers.

4.5 Implikationen

Um diese Sicherheitsanforderungen umzusetzen muss die Spezifikation zumindest im Punkt „Prüfung schreiben“ erweitert werden. Für Prüfungen als benotete Leistungskontrollen wird der Use-Case erweitert:

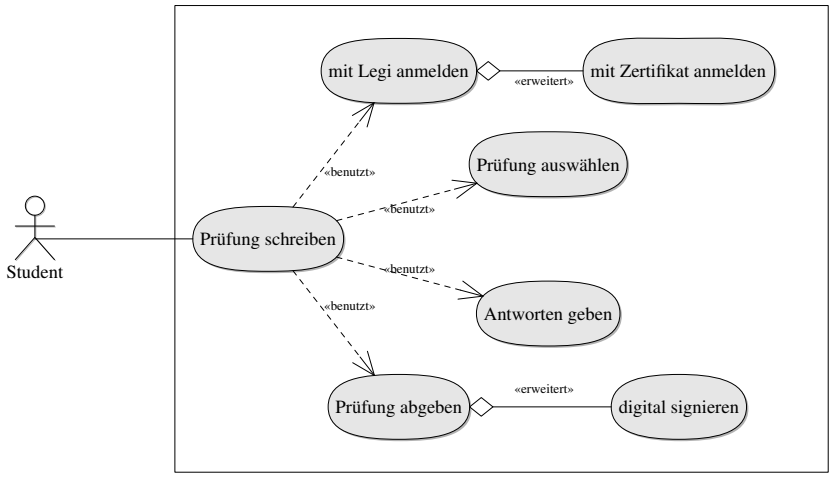


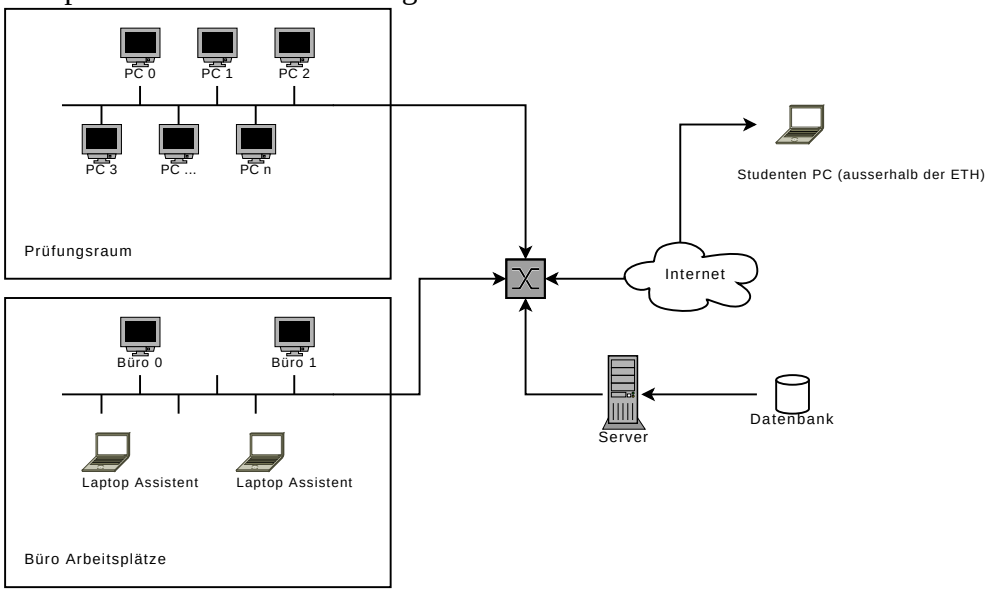
Abbildung 4.2: Erweiterter Use-Case

Wie man sieht, wurde die Anmeldung sowie die Abgabe um Zertifikate erweitert. Das Zertifikate auch im Use Case „Antworten geben“ involviert sind wird im UseCase nicht berücksichtigt, da es keiner aktiven Handlung des Akteurs bedarf. Alle anderen mit Zertifikaten zusammenhängenden Handlungen sollen bewusst erfolgen, vor allem der Prozess des Unterschreibens der Prüfung muss willentlich erfolgen, das verlangt das Gesetz.[3]

5 Design

5.1 Systemübersicht

Die Architektur des Systems wird weitgehend ähnlich sein zu den am Markt befindlichen Systemen. So besteht auch dieses System aus einem Server, Arbeitsplatz-PCs und dem Prüfungsraum sowie einer Netzwerk-Infrastruktur:



Die Komponenten entsprechen dabei den Bezeichnungen aus der Sicherheitsanalyse.

5.2 Datenmodell

Da das Datenmodell, wie später im Kapitel „Implementierung“ beschrieben, mithilfe von der Java Persistance Api umgesetzt wird, wird die Beschreibung mit UML Klassendiagrammen ausgeführt. Somit ist ein Mapping zum erstellten Programmcode einfacher nachzuvollziehen. Das Modell wird hier zur Veranschaulichung schrittweise aufgebaut. Die zentrale Entität, mit der die Beschreibung auch beginnt ist das „Item“ die Aufgabe.

5.2.1 Item / ItemRevisionGroup

Das „Item“ repräsentiert eine Aufgabe in einer Version im Fragenpool. Damit eine Versionierung der Aufgaben wie gefordert möglich wird, müssen sich Aufgaben zu Gruppen agglomerieren lassen. Diese Revisionsgruppen heissen im Modell „ItemRevisionGroup“. Bei allen UML-Darstellungen werden Getter- und Setter- Methoden aus Gründen der Übersichtlichkeit weggelassen. Das erste Modell ist also das Folgende:

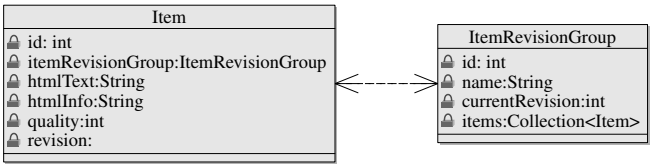


Abbildung 5.1: Item Model

Zwischen Item und ItemRevisionGroup besteht eine bidirektionale Relation zum Verwalten der Revisionen. Jedes Item hat eine aufsteigende Revisionsnummer und in der Revisionsgruppe ist die aktuelle Revisionsnummer verzeichnet.
Damit man möglichst viele Optionen zur Gestaltung der Frage zur Verfügung hat wurde entschieden, den eigentlichen Fragen-Text in HTML auszudrücken. Das hat vor allem auch den Vorteil, dass sich die Frage später leicht in ein XML Format umwandeln lässt, um den Austausch zwischen verschiedenen Systemen zu ermöglichen. Vgl. hierzu auch den QTI-Standard [1]. Die einzige Limitierung hierbei ist, dass das Dokument im HTML 3 Standard vorliegen muss. In der Praxis ergeben sich daraus aber keine allzu grossen Einbussen was Formatierungsmöglichkeiten anbetrifft. htmlText beinhaltet die eigentliche Frage, htmlInfo enthält zusätzliche Informationen zum Lösen der Aufgabe wie zum Beispiel die Punktezahl oder Tipps. Diese sollten jedoch so kurz wie möglich gehalten werden.

5.2.2 Subitem

Natürlich muss zu jeder Frage ein Satz Antworten existieren. Diese Antworten sollen in dieser Version erstmal ausschliesslich als Multiple-Choice Variante vorliegen. Da man aber nicht immer (vor allem nicht bei der Prüfungsdurchführung) Antwort-Objekte mit samt der Lösung verteilen will, wurde das Antwortobjekt „Subitem“ in zwei Teile zerlegt:

[1] IMS Global Learning Consorcium `http://www.imsproject.org/question/`

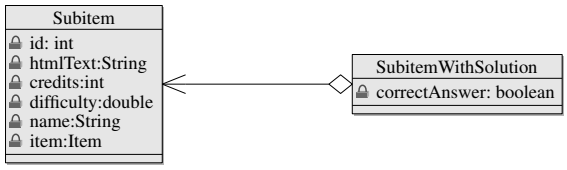


Abbildung 5.2: Subitem mit Lösung

Möchte man jetzt das Antwort-Objekt bei einer Prüfung herausgeben, so gibt man einfach das Objekt Subitem heraus, möchte man die gleiche Entität zur Korrektur verwenden, muss man „SubitemWithSolution“ verwenden.

5.2.3 Media

Da der Support für Grafiken innerhalb von Fragen grundlegender Teil der Anforderungen ist, müssen diese ebenfalls in der Datenbank gespeichert werden. Die Datenbank ist hier dem Dateisystem als Speicherort für Grafiken vorzuziehen, da es so einfacher ist mithilfe von Relationen die Übersicht über die Verwendung und Zugehörigkeit zu behalten um nicht mehr benutzte Bilder erkennen und löschen zu können. Auch für den Export von Prüfungsfragen ist es von Vorteil mit einem einfachen Query alle zu einer Frage gehörenden Grafiken extrahieren zu können.

5.2.4 Komplettes Datenmodell der Aufgaben

Setzt man alle bisherigen Modelle zusammen, so erhält man das komplette Datenmodell zum Abbilden einer Aufgabe im Prüfungs-System. Zu erwähnen ist noch, dass die Grafiken an die ItemRevisionGroup gebunden sind. Das hat sich bei der Implementierung als praktikabel herausgestellt, da man so alle Bilder zu einer Aufgabe gehörend in einer Datenstruktur zur Verfügung hat. Das Modell sieht also wie folgt aus:

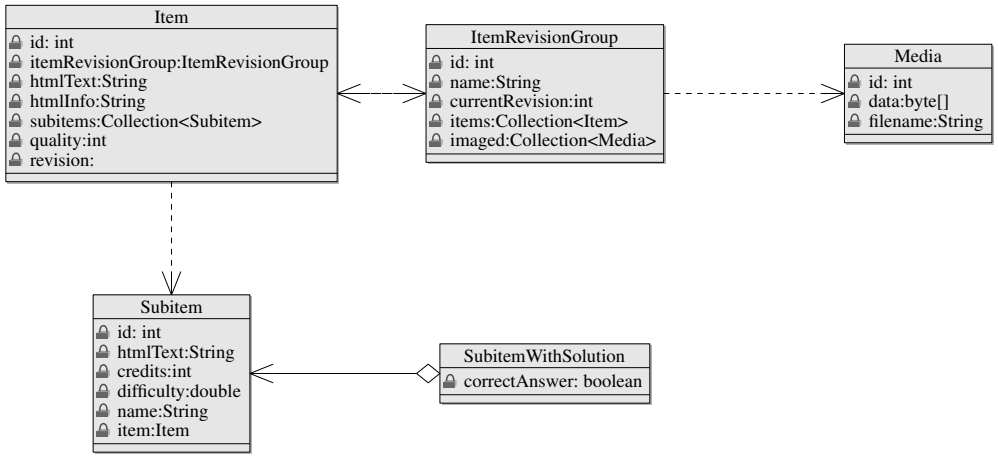


Abbildung 5.3: Datenmodell Aufgabe

5.2.5 Exam

Zu einer Prüfung gehört neben den Aufgaben auch ein Durchführungstermin sowie eine Liste der zugelassenen Studenten. Da dieses System nicht zur Kursadministration oder sogar zur Schulverwaltung eingesetzt werden soll, kann von einer zentralen Verwaltung der Studenten abgesehen werden. Darüber hinaus muss das Modell auch die Abgaben einzelner Fragen sowie das Prüfungsdokument abbilden können. Es ist folgendermassen modelliert:

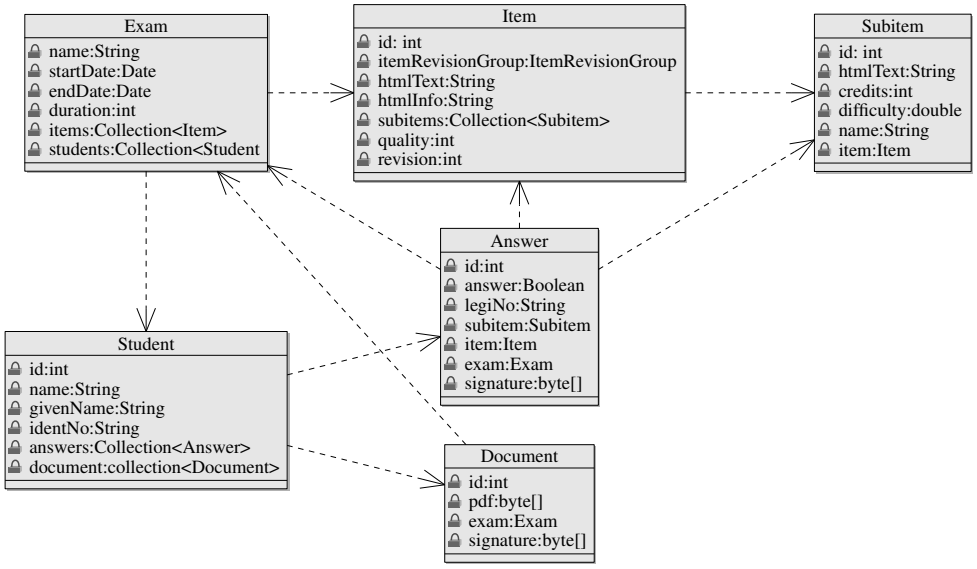


Abbildung 5.4: Exam Modell

5.2.6 User / Roles

Die Benutzerdaten des Systems sollen ebenfalls in der Datenstruktur abgebildet werden. Dafür sind vier Klassen vorgesehen: User, UserRole, Role, AcademicGroup. Mit Role werden die in der Spezifikation geforderten Benutzergruppen abgebildet. Ein User wird also einer Benutzergruppe zugeordnet, allerdings unter Bezug auf eine Gruppe im akademischen Sinne. Somit gibt es beispielsweise einen Assistenten „Markus“ in der Gruppe „Hinterberger“. Jeder Benutzer kann verschiedene Rollen in verschiedenen Gruppen annehmen. Was für Dozenten weniger häufig vorkommt als für Assistenten und vor allem aber für Hilfsassistenten. Das Datenmodell besteht aus:

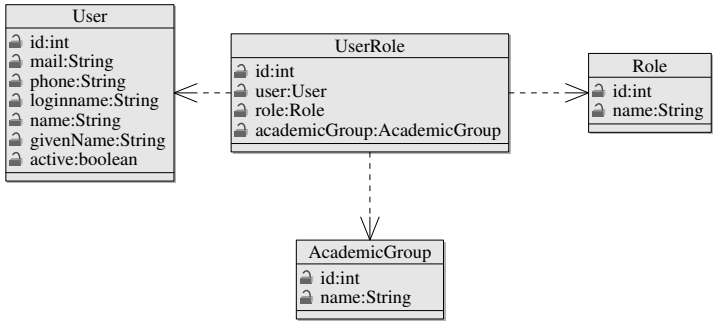


Abbildung 5.5: User Modell

5.2.7 SecurityMonitoredObject

Da ein wichtiges Ziel des Sicherheitskonzeptes die Geheimhaltung schützenswerter Daten ist, muss das System eine Entscheidungsgrundlage haben, wer welche Daten einsehen darf. Dies geschieht, in dem jede zu schützende Klasse von „SecurityMonitoredObject“ erbt zum Beispiel auch ItemRevisionGroup:

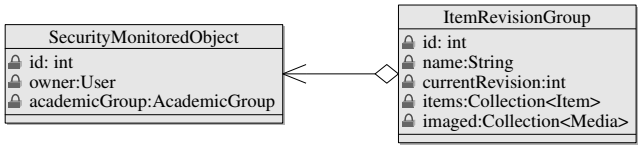


Abbildung 5.6: SecurityMonitoredObject

6 Implementierung

6.1 Wahl der Programmierumgebung

Um die in der Sicherheitsanalyse geforderten kryptographischen Protokolle ein- und umzusetzen muss zumindest für den Prüfungs-Client auf ein web-basiertes System verzichtet werden. Dies begründet sich vor allem darauf, dass eine automatische Signatur der gesendeten Daten nicht ohne Intervention des Benutzers möglich ist oder die entsprechenden Protokolle in Java-Script umgesetzt werden müssten. Beides ist nicht angezeigt und mir ist keine andere Möglichkeit bekannt dieses Problem zu lösen. Daher sind typische Web-System Sprachen wie PHP oder Ruby ausgeschieden. Deshalb wurde Java als Umgebung gewählt, da hier alle benötigten kryptographischen Protokolle als Libraries implementiert sind. Zusätzlich bekommt man die Plattformunabhängigkeit quasi gratis dazu. Somit kann das System unter Linux / Mac oder Windows ohne weitere Anpassungen betrieben werden.

6.2 Server

Bevor die eigentliche Implementierung des Server-Backends beschrieben wird, werden noch kurz Enterprise Java Beans vorgestellt. Enterprise Java Beans, kurz EJBs genannt, werden für diese Arbeit als zentrales Framework für die Implementierung des Serversystems eingesetzt. Unter [8] und [9] sind weitere Inforationen zu EJBs zu finden. Ein grosser Vorteil dieses Vorgehens ist, dass einem viele Fehler anfälligen Routinearbeiten vom Framework abgenommen werden. Für diese Arbeit von Interesse ist das OR-Mapping, DB-Pooling und Threadpooling. Das wird anschliessend im Einzelnen erklärt. Um die genannte Funktionalität bereitzustellen bedarf es eines Application-Servers. Es stehen dabei mehrere frei erhältliche Varianten zu Verfügung. Ich habe mich zu Gunsten des JBoss Application-Servers [1] entschieden, da sich dieser am besten für dieses Vorhaben eignet. Er ist sehr weit verbreitet, bietet eine gute Performance und viele Kleinigkeiten, die dieses System benötigt, sind meiner Meinung nach

[1] `http://www.jboss.org/`

leichter zu konfigurieren als bei der Konkurrenz. Dazu gehört die Anbindung an die Datenbank, die Login-Konfiguration, SSL-geschützte Client Verbindungen und nicht zu letzt auch die vergleichsweise kleinen Libraries für den Client. Mit etwa 4 MB fallen diese deutlich kleiner aus als beispielsweise beim Glassfish Application-Server von SUN, der dafür 16 MB benötigt. Für den JBoss gibt es ausserdem eine sehr gute Integration in die Entwicklungsumgebung Eclipse [2], die für dieses Projekt benutzt wurde.

6.2.1 OR-Mapping

Unter OR-Mapping, also Object Relational Mapping, versteht man die Abbildung der im Programm benutzten Datenobjekte in eine relationale Datenbank und umgekehrt. Hat man kein Framework zur Verfügung, so schreibt man meist selbst SQL-Statements in einer Load/Store Methode innerhalb der Datenklasse. Das hat aber gleich mehrere Nachteile. Zum einen ist solch ein Vorgehen langwierig, schlecht zu testen und damit immer eine mögliche Fehlerquelle, zum anderen legt man sich durch das Schreiben nativer Datenbank-Queries auf einen Datenbankhersteller fest. Ein Wechsel der Datenbank ist meist nur mit immensem Aufwand möglich. Verwendet man hingegen die Java Persistance API, wie sie von einem EJB-Container zur Verfügung gestellt wird, umgeht man die genannten Probleme elegant. Mithilfe von Annotationen spezifiziert man das Mapping, der Rest wird vom Framework erledigt. Dazu werden sogenannte Entity-Beans definiert. Das sind erstmal einfach POJOs (Plain-Old-Java-Objects) denen via Annotationen zusätzlich Semantik eingefügt wird. Das folgende Beispiel soll diesen Vorgang verdeutlichen:

```
@Entity
public class Item implements Serializable {

    @Id
    @GeneratedValue
    private int id;

    @ManyToOne
    private ItemRevisionGroup itemRevisionGroup;

    @ManyToOne
    private Module module;

    @OneToMany
    private Collection<Subitem> subitems =
```

[2] http://www.eclipse.org

```
        new ArrayList<Subitem>();

    @OneToMany
    private Collection<Comment> comments =
        new ArrayList<Comment>();

    private String htmlText;
    private String htmlInstructions;
    private int quality=1;
    private int revision;
}
```

Die hier gezeigte Item-Klasse aus dem Datenmodell (Getter und Setter wurden ausgeblendet) wird dann folgendermassen auf die Datenbank abgebildet:

```
+--------------------------------+
|          Table Item            |
+---------+----------------------+
| Integer | id                   |
| Integer | itemRevisionGroup_id |
| Integer | module_id            |
| Varchar | htmlText             |
| Varchar | htmlInstructions     |
| Integer | quality              |
| Integer | revision             |
+---------+----------------------+
```

Dabei werden dann alle Referenzen auf andere Objekte durch Fremdschlüssel in der Datenbank ersetzt, wie hier zum Beispiel module_id ein Ersatz für das Java-Feld module darstellt.
Das Interface zu diesem System wird durch den EntityManager zur Verfügung gestellt,

```
@Resrouce
EntityManager em;
```

der per Injection zur Laufzeit zur Verfügung gestellt wird. Überall dort, wo man einen EntityManager benötigt, kann er (wie oben gezeigt) eingebunden werden. Es gibt nun zwei grundlegende Möglichkeiten auf die Entity-Beans zuzugreifen. Einmal über die Id:

```
Item i = em.find(Item.class,5);
```

Oder man formuliert sich eine Query in EQL. EQL steht dabei für „Entity Query Language“ und ist SQL-Dialekt unabhängig. Man formuliert seine Query in

einer SQL ähnlichen Syntax aber allerdings auf Objektebene. Eine solche Query könnte wie folgt aussehen:

```
Query q = em.createQuery("SELECT i FROM Item AS i "+
                         "WHERE i.quality > :quality");
q.setParameter("quality", 2);
List<Item> itemList = q.getResultList();
```

Das Ergebnis der Query liegt später in der Datenstruktur itemList und kann weiter bearbeitet werden.

Objekte werden, wenn nicht anders angegeben, immer „Lazy“ geladen. Das heisst, es werden nur die Objekte der Funktionalität 1 geladen. Das ist zum Beispiel bei einer N:1 oder 1:1 Relation der Fall. 1:N oder N:M Relationen werden erstmal ignoriert. Das ist vor allem dann sehr wichtig, falls man Ringstrukturen in seinen Objekten-Graphen hat. Sonst würden diese zwangsläufig zu einer Endlosschleife führen oder den gesamten Datenbestand laden.
Das Speichern von Objekten geht genauso einfach. Es muss lediglich unterschieden werden ob das Objekt schon in der Datenbank vorhanden ist und nur ein Update erfahren soll oder ob ein neues Objekt der Datenbank hinzugefügt werden soll. Man benutzt daher

```
Item item = new Item();
item.setRevision(1);
em.persist(item);
```

um Objekt item als neues Objekt zu speichern und

```
Item item = em.find(Item.class,3);
item.setRevision(2);
em.merge(item);
```

um ein Update durchzuführen.

6.2.2 DB-Connection-Pooling

Bei Datenbank intensiven Applikationen, wie dieses System zweifelsfrei eines ist, können mehrere Verbindungen zum Datenbankserver in einem Pool gehalten werden, die der Anwendung dann im Bedarfsfall zur Verfügung stehen. Das ist von Vorteil, da das Aufbauen einer Verbindung inklusive Anmeldung am DB-Server zeitlich aufwändiger ist als eine bestehende Verbindung aus dem Pool zu recyceln. Auch dies wird vom Applikation-Server automatisiert. Bei allen mir bekannten Applikation-Servern kann die Pool-Grösse konfiguriert werden. [1]

6.2.3 Threadpooling und Server Interface

Unter Thread-Pooling versteht man, dass die für die Beantwortung eines Server-Requests benötigte Funktionalität in einem Thread zusammengefasst wird. Es existiert stets eine fixe Anzahl von diesen Threads zum Beispiel 25 in einem Pool. Für einen Client, der eine Anfrage stellt, wird ein Thread aus dem Pool entnommen und bearbeitet dann die Anfrage. Wird der Thread nicht mehr benutzt, kommt er zurück in den Pool. Erst wenn alle Threads benutzt sind, muss ein Client warten. Der Vorteil des Thread-Pooling ist, dass der Thread nicht erst erzeugt werden muss, was zeitaufwendig wäre. Die Threadpool Verwaltung ist, sollte man sie selbst implementieren, relativ aufwändig. Enterprise Java Beans bieten aber auch hier einen fertigen Mechanismus an, der einem das Thread-pooling abnimmt. Dazu implementiert man so genannte Session-Beans. Eine Session-Bean implementiert auf jeden Fall ein Interface. Entweder ein local oder remote Interface. Dort wird deklariert, welche Funktionen die Bean zur Verfügung stellt. Sei es lokal für andere Session-Beans oder entfernt für die Clients. In diesem Projekt sind die meisten Interaces Remote-Interfaces. Das AssistantBean Interface sieht beispielsweise so aus:

```
@Remote
    public interface AssistantBeanRemote {
    public int persistMedia(Media media);
    public int persistItem(Item item);
    public Module[] getMyModules();
    public Module[] getMyModules(AcademicGroup group);
    public ItemRevisionGroup
        getItemRevisionGroup(int id);
    public ItemRevisionGroupDTO[]
        getMyItems(AcademicGroup group, Module module);
    public ItemRevisionGroup
        persistItemRevisionGroup(ItemRevisionGroup itemrev);
}
```

Mehr als die deklarierten Methoden können am Server später auch nicht aufgerufen werden. Das bringt auf jeden Fall einen Sicherheitsgewinn, wenn das Server Interface klar spezifiziert ist. Es folgt dem „Principle of least Privileges“, welches vorschreibt, dass nicht mehr Informationen und Ressourcen zur Verfügung stehen sollen als für die Aufgabe nötig sind. [2]. Die @Remote Annotation drückt die Remote-Funktionalität aus. Das Gegenteil dazu wäre die @Local Annotation.
Die Sesseion-Beans dienen im EJB Konzept dazu die eigentliche Business-Logik zu implementieren. Je nach Applikationserver existiert ein, mehr oder weniger grosser, Bean-Pool in dem viele Instanzen einer Bean vorhanden sind. Es gibt dabei zwei Arten von Session Beans: stateless und statefull Beans.

Stateless Beans können keine verbindungsbezogenen Daten speichern und können überall dort eingesetzt werden, wo dies nicht nötig ist. Statefull Beans sind dagegen zustandsbehaftet. Jede Verbindung bekommt eine dieser Beans zugeordnet. Die Bean ist dabei beständig, solange die Verbindung aufrecht erhalten wird. Ist der Beanpool erschöpft, so können diese Beans auch passiviert werden um so Platz für neue und aktive Verbindungen zu schaffen. Aus drei Gründen wurde der Server aber fast ausschliesslich mit Stateless Beans implementiert. Zum einen gibt es nach einem Verbindungsverlust Probleme wieder an die Stateful Bean zu binden, zum anderen belasten Stateless Beans das System mit weniger Overhead, was bei vielen gleichzeitigen Benutzern ein Vorteil ist und zum Dritten können sehr leicht später WebServices angeboten werden, da diese ja ohnehin nur stateless sein müssen.
Eine einfache Session-Bean sieht zum Beispiel folgendermassen aus:

```
@Stateless
public class TestBean implements TestBeanRemote {

        public Item getItembyId(int id) {
                return em.find(Item.class,id);
        }
}
```

6.2.4 Rechteverwaltung

Natürlich ist die gerade eben vorgestellte Bean sehr unsicher, da sie völlig ungeprüft Aufgaben herausgibt sobald man die Datenbank Id kennt. Um das zu verhindern stellt das EJB System ein ausgeklügeltes Rechtemanagement zur Verfügung. So muss sich ein Client am Server anmelden, bevor dieser irgend eine Bean aufrufen kann. Das kann anonym erfolgen oder mit Benutzername und Passwort. Der Benutzername und das Passwort werden dann mit einem Authentifizierungsmechnaismus überprüft, der frei gewählt werden kann. In unserem Falle werden die Clients natürlich gegen die Datenbank authentifiziert. Das heisst, sobald sich ein Client mit dem Server verbunden hat, weiss der Server, welche Rollen dem Client zugeordnet sind. Dies kann gleich beim Methodenaufruf überprüft werden.

```
@Stateless
public class TestBean implements TestBeanRemote {

    @RolesAllowed({RoleNames.Assistant})
    public Item getItembyId(int id) {
        return em.find(Item.class,id);
    }
```

```
}
```

RoleNames ist eine selbst definierte Klasse, die die Namen der Rollen als static String abspeichert. Der EJB-Container überprüft nun bei Aufruf ob sich der Rollenname der Anmeldung mit dem spezifizierten deckt. Ist dies nicht der Fall, wird die Ausführung unterbunden und eine SecurityException ausgelöst. Diesen Ansatz nennt man Declarative Security.
Das reicht natürlich für die gezeigte Bean noch nicht aus, da ja auch noch von Interesse ist in welcher Gruppe der Client die Rolle Assistent besitzt. Das kann man aber leicht selbst überprüfen:

```
@Stateless
public class TestBean implements TestBeanRemote {
    @Resource
    Context ctx;
    @RolesAllowed({RoleNames.Assistant})
    public Item getItembyId(int id) {
        if(ctx.getCallerPrincipal.getName().
            equals("Juergen"))
            return em.find(Item.class,id);
        return null;
    }
}
```

Mithilfe des aktuellen Contextes, auf den man wie gezeigt zugreifen kann, bekommt man die Datenstruktur, welche die Login-Informationen des Clients enthält. Im Beispiel wird der Loginname überprüft, man hat aber auch auf die Rollen und weitere Informationen Zugriff.

6.3 Clients

Die Clients werden als Standard Java Applikation mit einer Swing GUI entwickelt. Die Entscheidung hierzu ist vor allem durch die besseren Kryptographie-Libraries begründet. Da die Aufgaben alle als HTML verfasst sind, ist es von grundlegender Notwendigkeit HTML im Client zu rendern und darzustellen. Das Swing GUI Kit hat gerade für HTML-Darstellungen einen guten Support.

6.3.1 Rendering HTML

Die Swing GUI hält dafür ein Element namens JEditorPane bereit. Diese Pane ist vor allem zum Editieren von formatierten Texten gedacht, kann aber auch nur zum Anzeigen benutzt werden. Die einfachste Anwendung der JEditorPane sieht folgendermassen aus:

```
@Entity
public class Test extends JFrame {

    public Test() {
        this.setSize(800,600);
        JEditorPane pane = new JPanel("http://www.google.ch");
        this.getContentPane.add(pane);
        this.setVisible(true);
    }

    public static void main(String[] args) {
        Test t = new Test();
    }

}
```

Man erhält ein neues Fenster und die Google-Webseite wird angezeigt. Es gibt hier allerdings die Einschränkung, dass HTML nur bis zur Version 3.0 unterstützt wird und kein CSS Support vorhanden ist. Auch erhält man so nur eine Darstellung der Seite, die Funktionalität ist nicht vorhanden. Das heisst, es funktionieren keine Links oder Formulare, dafür wäre viel mehr Code nötig. Für die Client-Applikaitonen ist das aber schon ausreichend. Ein Problem ist, dass bei den HTML-Daten, die zum Beispiel eine Aufgabe beschreiben, die Bilder nicht wie gewöhnlich bei HTML über eine URL erreichbar sind. Zum einen soll nicht auch noch ein Web-Server laufen, der die Bilder zur Verfügung stellt, zum anderen verbietet sich dieses Vorgehen schon alleine durch den Sicherheitsaspekt. Die Herausgabe von Grafiken darf nur kontrolliert und autorisiert erfolgen, da Aufgaben unter Umständen vollständig aus Grafiken bestehen könnten. Ich musste also die JEditorPane dazu bringen, die nötigen Bilder aus einer separaten Datenstruktur zu übernehmen und anzuzeigen. Das Java-Framework kommt einem bei solchen Änderungen entgegen, da alles sehr modular aufgebaut ist. So ist zum Beispiel die HTML Funktionalität des JEditorPanes in eine Klasse HTMLEdtitorKit gekapselt. Man kann dazu einfach von HTMLEditorKit erben und die Behandlung der „<img>“ Tags ersetzen. Die Tags sehen beim Prüfungs-System so aus:

```
<img id=10/>
```

Wobei die id dem Primary-Key der Tabelle Media entspricht. So sehen die abgeleiteten Klassen des HTMLEditorKit aus:

```
@Entity
public class EHTMLEditorKit extends HTMLEditorKit {
   private Item item;
```

```
   private static final long serialVersionUID = 1L;

   public EHTMLEditorKit(Item item) {
      this.item = item;
   }

   public ViewFactory getViewFactory() {
      return new EHTMLViewFactory(item);
   }

}

public class EHTMLViewFactory extends HTMLEditorKit.HTMLFactory {

   private Item item;

   public EHTMLViewFactory(Item item) {
      super();
      this.item = item;
   }

   private ImageIcon findImage(String imagekey) {
      for(Media m :item.getItemRevisionGroup().getMedia()) {
         if(m.getId() == Integer.parseInt(imagekey)) {
            return new ImageIcon(m.getImageData());
         }
      }
      return new ImageIcon();
   }

   public View create(Element elem) {

      if( elem.getName().equals("img") ){
         String imagekey = elem.getAttributes().
                getAttribute(HTML.Attribute.SRC).toString();
         View view =  new EHTMLComponentView(elem,findImage(imagekey));
         return view;
      }
      return super.create(elem);
   }
}

public class EHTMLComponentView extends ComponentView {

   private ImageIcon image;

   public EHTMLComponentView(Element elem, ImageIcon image) {
      super(elem);
```

```
      this.image = image;
   }

   protected Component createComponent() {
      return new JLabel(image);
   }
}
```

Die abgeleitete Klasse EHTMLEditorkit kann man jetzt einfach dem JEditorPane als EditorKit laden. Der folgende Code zeigt das. Man beachte, dass die Datenstruktur, in diesem Falle die HashMap dem Konstruktor des EHTMLEditorKit übergeben wird. Darin müssen dann alle Bilder, der zu rendernden HTML-Darstellung enthalten sein. Fehlt ein Bild, wird es lediglich nicht angezeigt.

```
infoEditor.setEditorKit(new EHTMLEditorKit(item));
infoEditor.setContentType("text/html");
```

6.3.2 Admin-Client

Der Admin-Client ist als klassische Java GUI Applikation ausgeführt. Als GUI Toolkit wird Swing verwendet. Die Main-Methode befindet sich in der Klasse AdminClient und ist selbst von JFrame abgeleitet. Damit stellt sie das zentrale Fenster der Anwendung zur Verfügung. In dieser Klasse wird das Menü aufgebaut und die Verbindung zum Server hergestellt. Auch das Login wird gleich bei der Initialisierung durchgeführt. Schlägt die Verbindung oder das Login fehl, beendet sich die Software mit der entsprechenden Fehlermeldung.

Multi-Language

Das System ist clientseitig durchgehend auf Mehrsprachigkeit ausgelegt. Dazu werden alle Strings, die in einer Anzeige erscheinen, in eine Properties-Datei ausgelagert und mit einer eindeutigen Identifikation referenziert. So ist zum Beispiel der Titel, der im Applikationsfenster erscheint, in der Properties-Datei unter ADMIN_CLIENT_TITLE zu finden. Dabei steht der erste Teil der Identifikation für den Klassennamen. Um die Applikation mit einer anderen Sprachumgebung zu nutzen, muss lediglich das properties-File in die gewünschte Sprache übersetzt werden. Die entsprechenden Dateien liegen im Package ch.ethz.inf.sioux.resources und heissen Language_local.properties, wobei local hierbei das Kürzel der jeweiligen Sprache darstellt. Also „de“ für Deutsch oder „en“ für Englisch. Welches Local Verengung findet ist im File Configuration_all.properties durch den Parameter LANG geregelt.

Server Verbindung

Die Verbindung zum Server erfolgt direkt auf EJB-Ebene. Die Kommunikation erfolgt dabei, wenn man den JBoss einsetzt über RMI. Als ersten Schritt muss man sich am Server anmelden. Dazu erzeugt man sich eine Instanz von Initial-Context. Dem Context werden noch Parameter mitgegeben zum Beispiel die URL des Servers und die Authentifizierungsdaten des Benutzers. Wenn SSL Verschlüsselung gewünscht wird, kann dies auch angegeben werden.

```
Properties properties = new Properties();
properties.setProperty(Context.SECURITY_PRINCIPAL,
        "peterhe");
properties.setProperty(Context.SECURITY_CREDENTIALS,
        "abc123");
properties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
        org.jboss.security.jndi.JndiLoginInitialContextFactory);
properties.setProperty(Context.URL_PKG_PREFIXES,
        org.jboss.naming.client);
properties.setProperty(Context.PROVIDER_URL,
        jnp://localhost:1099);
try {
        Context ctx = new InitialContext(properties);
        TestBeanRemote testBean = (TestBeanRemote)
                ctx.lookup("TestBean/remote");
}
catch (Exception e) {
        e.printStackTrace();
}
```

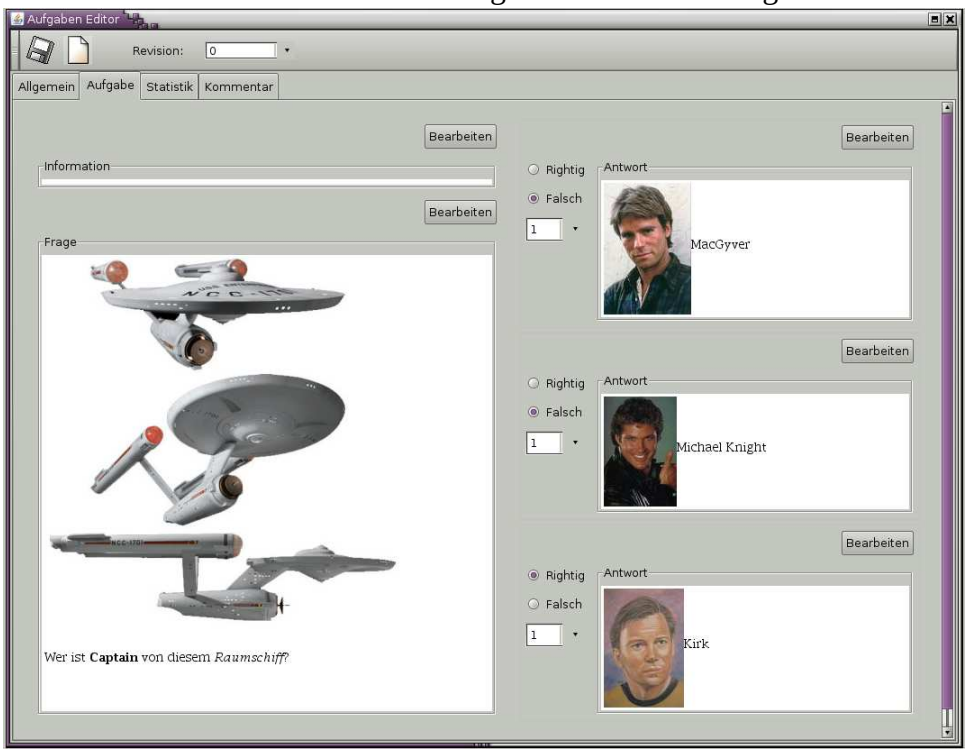
In diesem Beispiel meldet sich der Benutzer „peterhe“ mit Passwort „abc123“ am Server an. Der JNDI Name, der beim Lookup benutzt wird, ist abhängig von benutzten Applikation-Server. Im Falle des JBoss ist es, wenn nicht anders angegeben, Klassen-Name/remote.
Wird keine Exception ausgelöst, ist die Verbindung erfolgreich zustande gekommen und kann benutzt werden. Das heisst aber noch nicht, dass auch der Name und oder das Passwort akzeptiert wurden. Ist das Login fehlgeschlagen, ist man einfach als anonymer Benutzer am System angemeldet.

Desktop

Der Admin-Client ist als Mehrfenster-Applikation ausgelegt. Das heisst, es gibt eine Arbeitsfläche auf der mehrere Prüfungen oder Aufgaben gleichzeitig zur Bearbeitung geöffnet sein können. Dazu habe ich als zentrales GUI-Element im Client eine JDesktop-Pane verwendet, in der die einzelnen Fenster gehalten werden.

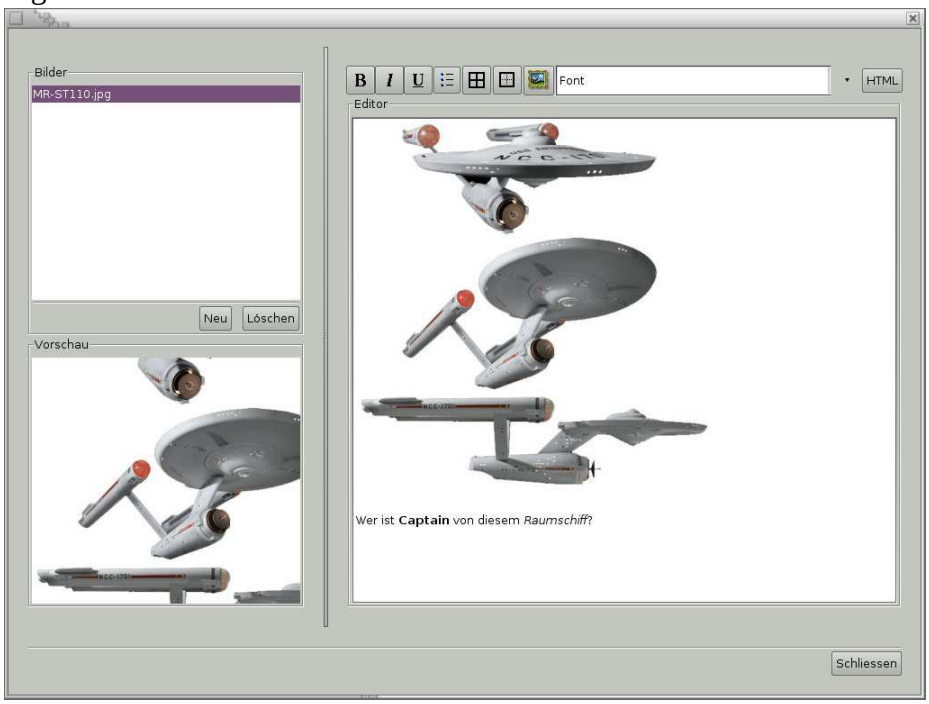
Item Editor

Um die Aufgaben zu bearbeiten habe ich das GUI Modul Item-Editor geschrieben. Es ermöglicht das Verändern aller Aspekte einer Aufgabe. Dazu wurde das GUI mit vier Registerkarten versehen. Auf der ersten Karte kann man die Metadaten der Aufgabe bearbeiten, so zum Beispiel den Namen der Aufgabe oder die geschätzte Schwierigkeit. Ein weiters Tab steht für statistische Auswertungen zur Verfügung, noch eine Tab existiert für die Verwaltung von Kommentaren und schliesslich gibt es noch die Tab, welche den eigentlichen Inhalt der Aufgabe enthält. Dort bekommt man eine Darstellung der Aufgabe, wie sie auch im Exam-Client zu sehen wäre, mit der Ausnahme, dass alle erfassten Subitems sichtbar sind. In der echten Prüfung wären nur fünf davon zugänglich. Alle HTML-Darstellungen können bearbeitet werden. Dazu öffnet sich, nach einem Klick auf den jeweiligen Editier-Button, der HTML-Editor wie im nächsten Kapitel beschrieben.
Sobald man sich entscheidet den aktuellen Stand auf dem Server zu speichern wird eine neue Revision der Aufgabe erstellt. Dazu wird man aufgefordert einen Kommentar bezüglich den durchgeführten Änderungen anzugeben. Anschliessend ist der neue Stand der Aufgabe am Server verfügbar.



HTML Editor

Der HTML-Editor ist zum Bearbeiten und Erstellen sämtlicher im System vorgesehen HTML-Darstellungen. Dazu besteht dieses GUI-Element aus einer JEditorPane in der, je nach Auswahl, entweder HTML in der Source-Form editiert wird oder aber auch direkt in der gerenderten Form editiert werden kann. Es gibt für das Editieren in gerenderter Darstellung einen Satz Funktionen, die man auch von einem normalen Texteditor her gewöhnt ist. Dazu zählt zum Beispiel der Underline, Italic oder Cut/Paste Button. Des Weiteren wird im HTML-Editor auch die Bilder-Liste der einzubindenden Grafiken verwaltet. Die Grafiken können von einem lokal gespeicherten Bild im Format JPEG, PNG, GIF oder Bitmap geladen und am Server gespeichert werden. Auf Wunsch können diese Bilder an beliebiger Stelle im HTML-Dokument eingefügt werden.



Exam Editor

Der Exam-Editor ist ähnlich aufgebaut wie der Aufgaben-Editor. Auch dieser verfügt über Registerkarten zur Navigation. Neben einer Registerkarte für die Metadaten zum Exam, beispielsweise Datum und Prüfungsart, stellt die GUI für jede im Exam vorkommende Aufgabe ein Tab zur Verfügung. Innerhalb der Aufgabenkarten können dann die Antworten zu den Fragen ausgewählt

werden, die in der Prüfung sichtbar sein sollen.

6.3.3 Exam-Client

Der Exam-Client ist ebenfalls als klassische Java-GUI Applikation ausgeführt. Der interne Aufbau entspricht aber eher einem endlichen Automaten, wie in der Grafik dargestellt:

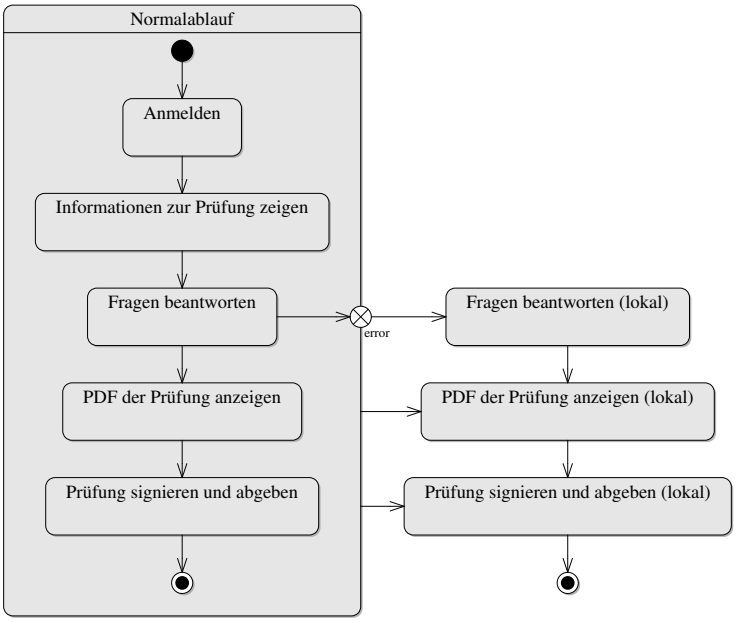


Abbildung 6.1: Client State Diagramm

Es gibt dabei, wie im Diagramm zu sehen, zwei grosse „Überstates“. Der Normalablauf, bei dem alle Systeme nominal funktionieren und der Fehlerfall, bei welchem die Kommunikation zum Server unterbrochen wird.

Normalablauf

Die Zustände entsprechen im Wesentlichen den in vorherigen Kapiteln gezeigten Use Cases für das Prüfungsfrontend. Bei dem letzten State „Prüfung signieren und abgeben“, kann das Signieren auch ausgelassen werden, sollte es sich nur um eine Probeprüfung mit reiner Legi basierten Autentifikation handeln.

Fehlerfall

Wie man in dem Diagramm sieht, ist der erste mögliche Übergang vom Normal- in den Fehlermodus erst ab dem State „Fragen beantworten“ möglich. Das ist auch nachvollziehbar, denn erst zu diesem Zeitpunkt ist die gesamte Prüfung im Speicher der Arbeitsstation des Prüflings vorhanden. Fällt die Verbindung zum Server vorzeitig aus, kann die Prüfung nur noch auf Papier durchgeführt werden. Der gesamte Fehlermodus kann generell nur dann zur Verfügung stehen, wenn es eine Möglichkeit gibt, einen mobilen Datenträger an die Arbeitsplätze anzuschliessen. Im Falle der digitalen Schlüssel auf dem USB-Stick ist dies aber ohnehin der Fall. Ist die Verbindung zum Server gestört, werden alle weiteren Antworten und Abgaben auf diesen lokalen Datenträger geschrieben. Am Ende der Prüfung müssen diese Datenträger dann in geeigneter Weise „eingesammelt“ werden. Dazu reicht auch das Kopieren der entsprechenden Dateien auf ein vertrauenswürdiges Laptop zum Beispiel auf das Gerät des Assistenten. Die so gewonnenen Daten können aber trotzdem als valid und authentisch betrachtet werden, da sie über die gleichen Signaturmerkmale wie die am Server gespeicherten Daten verfügen.

Zugriff auf die Zertifikate

Am Prüfungs-Client werden beide Teile des Schlüssels benötigt. Der Public-Key wird dem Server bei der Anmeldung mitgeteilt, der Private-Key wird zum Signieren benötigt. Beide Schlüsselteile sind auf dem USB-Stick untergebracht. Je nach Betriebssystem gibt es verschiedene Vorgehensweisen um dies Daten einzulesen:

Windows Bei Windows basierten Betriebssystemen werden einfach die Laufwerke „C-Z“ auf ihre Verfügbarkeit zyklisch überprüft. Tritt ein neues Laufwerk hinzu, in Form des USB-Sticks, wird dort automatisch nach dem entsprechenden Schlüssel gesucht. Das bietet für den Benutzer den Vorteil, dass die Schlüssel automatisch direkt nach dem Einstecken des USB-Sticks geladen werden.

Linux/UNIX Unter Linux oder UNIX im Allgemeinen ist ein solcher Vorgang nicht implementiert. Da sich der Einhängepunkt eines solchen Mediums sehr frei konfigurieren lässt, ist ein einfaches Überprüfen auf Vorhandensein nicht ohne weiteres möglich. Auch gibt es hier grosse Unterscheide zwischen den Distributionen. Daher wird als Übergangslösung dem Benutzer erstmal ein

Datei-Auswahldialog angeboten, in welchem er selber den Pfad des Schlüssels angeben muss.

Darstellung der Prüfung

Zur Darstellung der Prüfung sind im Client ebenfalls Registerkarten verwendet worden. Das HTML-Rendering erfolgt dabei genau wie beim Admin-Client, also wieder über eine JEditorPane. Pro Frage gibt es eine Registerkarte, der Fragentext befindet sich in der linken Bilschirmhälfte, die Antworten sind vertikal über der rechten Bildschirmhälfte verteilt. Diese Anordung wurde einer komplett vertikalen Anordung vorgezogen, da Computer-Bildschirme i.d.R. breiter als hoch sind. Diese Eigenschaft wird durch die Einführung von WideScreens noch verstärkt. Somit ergibt sich eine optimalere Bildschirmausnutzung.

Sollten die Aufgaben und Antworten grösser sein als auf dem Bildschirm darstellbar, so ist die Möglichkeit des Scrollens gegeben. Dies sollte aber unter allen Umständen bei der Erstellung der Aufgaben vermieden werden, da es auf den Benutzer ablenkend wirken kann.

7 Aufbau der Infrastruktur

7.1 Client

Eine Installation im eigentlichen Sinne der Software am Client ist nicht nötig. Es muss lediglich ein Java Runtime Environment (JRE) zur Verfügung gestellt werden. Sollte dies nicht vorhanden sein, kann es unter [1] bezogen werden. Die Version sollte nicht unter 6 liegen, denn damit habe ich die Software entwickelt und gestestet. Das Betriebssystem von dem Admin-Client und Exam-Client kann wahlweise Windows oder Linux sein.

7.2 Server

Die Installation der serverseitigen Software wird hier am Beispiel von Debian oder Ubuntu Linux dargestellt.

7.2.1 Java

Als erste Komponente muss ein Java Development Kit (JDK) installiert werden. Dieses kann direkt aus dem Repository von Debian oder Ubuntu installiert werden:

```
bash$ sudo apt-get install sun-java6-jdk
```

Als nächstes sollte noch getestet werden ob das System auch wirklich die soeben installierte Java-Umgebung benutzt. Die Eingabe von

```
bash$ java -version
```

sollte folgendes anzeigen:

```
java version "1.6.0_03"
Java(TM) SE Runtime Environment (build 1.6.0_03-b05)
Java HotSpot(TM) Server VM (build 1.6.0_03-b05, mixed mode)
```

Damit ist die Java Installation abgeschlossen.

[1] `http://java.sun.com/javase/downloads/index.jsp`

7.2.2 MySQL Server

Auch der MySQL Server kann direkt aus dem Repository installiert werden:

```
bash$ sudo apt-get install mysql-server
```

Während der Installation wird das MySQL-Root-Passwort abgefragt. Dieses sollte mindestens sechs Zeichen lang sein und nicht dem System-Root Passwort entsprechen.
Als erstes wird eine Datenbank benötigt, in der der Sioux-Server seine Tabellen ablegen kann. Diese kann mit

```
bash$ mysqladmin -u root -p create sioux
```

angelegt werden. Um diese Datenbank nutzen zu können, bedarf es auch noch einem User-Account. Um diesen einzurichten, muss man sich als Root-Benutzer an der Datenbank anmelden. Die ganze Befehlsabfolge lautet:

```
bash$ mysql -u root -p
Enter password:
...
mysql> use mysql;
mysql> insert into user (Host, User, Password) values
('localhost','sioux',password('sioux'));
mysql> insert into db (Host, User, Db, Select_priv, Insert_priv,
    Update_priv, Delete_priv, Create_priv, Drop_priv, Grant_priv,
    References_priv, Index_priv, Alter_priv) values ('localhost',
    'sioux','sioux','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> quit

bash$ mysqladmin -u root -p flush-privileges
```

Um das Ergebnis zu testen, sollte man kurz versuchen sich als Benutzer „sioux“ anzumelden:

```
bash$ mysql -u sioux -p
mysql> use sioux;
```

Funktioniert dies ohne Fehlermeldung, ist der DB-Server korrekt installiert.

7.2.3 JBoss Application Server

Der JBoss Application Server sollte von der `www.jboss.org` herunter geladen werden. Man kann auch den direkten Link zu einem der Mirror verwenden [1]. Da der Server nicht mit root-Rechten laufen sollte, muss das Installationsverzeichnis dem User gehören unter dem der Server später läuft:

```
bash$ cd /tmp
bash$ wget http://heanet.dl.sourceforge.net/sourceforge/\
jboss/jboss-4.2.3.GA-jdk6.zip
bash$ unzip jboss-4.2.3.GA-jdk6.zip
bash$ cd /opt
bash$ sudo mv /tmp/jboss-4.2.3.GA .
bash$ sudo chown peterhe jboss-4.2.3.GA -R
```

Im letzen Befehl ist natürlich der Benutzername „peterhe“ durch den am System angemeldeten Benutzernamen auszutauschen.
Der nächste Schritt ist den JBoss so zu konfigurieren, dass er auf die Datenbank zugriff bekommt. Dazu muss zuerst der Datenbank-Treiber für MySQL herunter geladen werden. Dieser ist unter

`http://www.mysql.com/products/connector/j/` zu bekommen.

```
bash$ cd /tmp
bash$ wget http://mirror.switch.ch/ftp/mirror/mysql/Downloads/\
Connector-J/mysql-connector-java-5.1.6.tar.gz
bash$ tar xzvf mysql-connector-java-5.1.6.tar.gz
bash$ cd mysql-connector-java-5.1.6/
bash$ cp mysql-connector-java-5.1.6-bin.jar \
 /opt/jboss-4.2.3.GA/server/default/lib
```

Damit ist der Treiber installiert. Jetzt muss die Konfigurationsdatei für den MySQL Server noch von den Beispielvorlagen kopiert und angepasst werden:

```
bahs$ cd /opt/jboss-4.2.3.GA
bash$ cp docs/examples/jca/mysql-ds.xml server/default/deploy/
```

Die Datei mysql-ds.xml sollte nach den Änderungen folgenden Inhalt haben:

[1] `http://heanet.dl.sourceforge.net/sourceforge/jboss/jboss-4.2.3.GA-jdk6.zip`

```
<?xml version="1.0" encoding="UTF-8"?>

<datasources>
  <local-tx-datasource>
    <jndi-name>MySqlDS</jndi-name>
    <connection-url>
jdbc:mysql://localhost:3306/sioux
    </connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>sioux</user-name>
    <password>sioux</password>
    <exception-sorter-class-name>
    org.jboss.resource.
adapter.jdbc.vendor.
MySQLExceptionSorter
    </exception-sorter-class-name>
    <metadata>
       <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

7.2.4 Login Konfiguration

Damit das Benutzerlogin für das Admin-Frontend funktioniert, muss dem JBoss-Server noch konfiguriert werden aus welchen Tabellen er die Benutzer und Passwort-Informationen herauslesen soll:
In der Datei „login-config.xml“ unter /opt/jboss-4.2.3.GA/server/default/conf kann dies eingestellt werden. Dazu fügt man eine neue Application-policy hinzu, und zwar an der Stelle, wo auch schon andere Application-Pilocies definiert sind. Diese sollte so aussehen:

```
 <application-policy name = "sioux">
       <authentication>
          <login-module
code = "org.jboss.security.auth.spi.DatabaseServerLoginModule"
              flag = "required">
              <module-option
name = "unauthenticatedIdentity">guest</module-option>
              <module-option
```

```
name = "dsJndiName">java:/sioux</module-option>
             <module-option
name = "principalsQuery">
SELECT l.password
FROM User AS u, UserWithLogin AS l
WHERE u.id=l.id
AND u.loginname=?</module-option>
             <module-option
name = "rolesQuery">
SELECT r.name , ""
FROM Role AS r,
User AS u,
UserRole AS ur
WHERE r.id = ur.role_id
AND ur.user_id = u.id
AND u.activated = 1
AND u.loginname=?</module-option>
          </login-module>
       </authentication>
    </application-policy>
```

Damit ist der Server fertig konfiguriert, und kann auf die Benutzerdaten und Rollen auf der Datenbank zugreifen.

7.2.5 Erster Start

Um den Server in Betrieb zu nehmen wird die Datei sioux-server.jar in den Deploy-Ordner des Application Servers kopiert und der Server gestartet. Dabei werden automatisch die Tabellen in der Datenbank angelegt. Damit man sich am Admin-Client anmelden kann, müssen einige wenige inertiale Daten in die Datenbank eingefügt werden. Dazu gehört unter anderem auch die Login-Daten für den Benutzer „admin“.

```
INSERT INTO ‘Role‘ (‘id‘, ‘name‘) VALUES
(1, ’Administrator’),
(2, ’Academic’),
(3, ’Assistant’),
(4, ’TeachingAssistant’),
(5, ’Student’);
```

```
INSERT INTO ‘User‘ (‘id‘, ‘mail‘, ‘phone‘,
‘loginname‘, ‘name‘,
‘givenName‘, ‘activated‘) VALUES
(1, NULL, NULL, ’admin’, ’Administrator’,
’System’, ’1’);

INSERT INTO ‘UserRole‘ (‘id‘,
‘academicGroup_id‘, ‘user_id‘,
‘role_id‘) VALUES
(1, NULL, 1, 1);

INSERT INTO ‘UserWithLogin‘ (‘id‘,
‘password‘) VALUES
(1, ’admin’);
```

7.2.6 Log-Files

Der JBoss Server wird standardmässig mit maximalem Logging ausgeliefert. In unserem Falle heisst das aber auch, dass sogar die Inhalte der Datenbank-Requests angezeigt und im Logfile gespeichert würden, was natürlich alles andere als wünschenswert ist. Daher muss die Datei „/opt/jboss-4.2.3.GA/servers/default/conf/jboss-log4j.xml“ angepasst werden. Es muss dazu einfach der Eintrag

```
<category name="org.hibernate">
<priority value="WARN"/>
</category>
```

hinzugefügt werden.

8 Evaluation

8.1 Performance-Messung

Zum Messung der Serverleistung habe ich ein Testprogramm implementiert, dass die gleichzeitige Benutzung des Prüfung-Clients, simuliert. Dazu verwendet es die gleichen Mechanismen wie der Prüfungs-Client um sich anzumelden, Prüfungen zu laden und Antworten abzugeben. Alle Messungen wurden auf einem Dual-Core Laptop mit 4 GB Ram durchgeführt. Zwei zeitkritische Vorgänge wurden gemessen:

8.1.1 Übertragung der Prüfung zum Arbeitsplatz

Bei Beginn einer Prüfung ist damit zu rechnen, dass die Kandidaten nahezu gleichzeitig die Prüfungen vom Server anfordern. Um die Messung durchzuführen, wurde ein Exam mit einer Grösse von 2MB erstellt. 2MB sind eher die Obergrenze und dürften nur mit extrem grafiklastigen Prüfungen erreichbar sein. Die folgende Grafik zeigt die Messergebnisse dieses Versuches an:

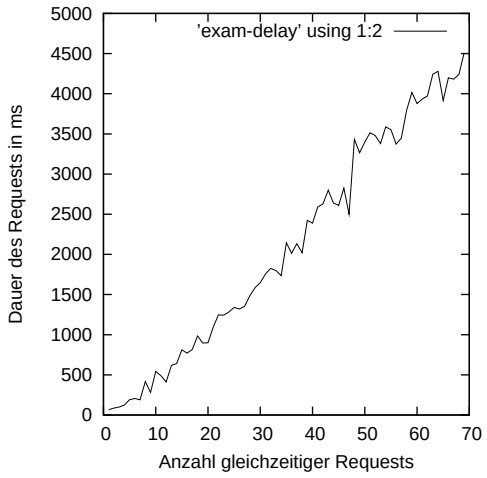


Abbildung 8.1: Messung der Verzögerung beim Laden der Prüfung

Man sieht hierbei den relativ linearen Anstieg der Übertragungsdauer. 50 gleichzeitige Requests bei je etwa 2,5 Sekunden Verzögerung sollten aber völlig ausreichend sein, wenn nicht mehr als 200 Studenten gleichzeitig geprüft werden sollen. Diese Zahl begründet sich in der Annahme, dass durch die Eingabe der Legi-Nummer und der Auswahl der Prüfung eine zeitliche Verteilung entsteht, die in nicht mehr als 50 gleichzeitigen Anfragen resultiert. Natürlich könnten weit mehr Studenten gleichzeitig geprüft werden, wenn man diese Streuung künstlich etwas verstärkt. So würde beispielsweise eine Legi-Kontrolle am Eingang oder die Ausstellung der digitalen Schlüssel das gewünschte Ergebnis erbringen.

8.1.2 Abgabe der Antworten

Da der Prüfungs-Client eine abgegebene Antwort sofort an den Server überträgt und auf dessen Rückmeldung wartet ergibt sich eine Reaktionsverzögerung beim Geben der Antworten. Diese sollte etwa 500 ms nicht übersteigen. Ich würde ein System mit längerer Reaktionszeit als unangenehm empfinden. Durch folgende Messung soll diese Grenze gefunden werden:

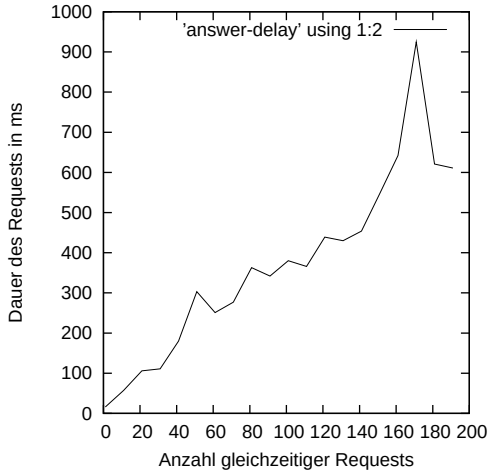


Abbildung 8.2: Messung der Verzögerung bei der Antwortgabe

Die 500 ms werden bei etwa 150 gleichzeitigen Requests erreicht. Da die Antworten erstens sporadisch und fast uniform verteilt gegeben werden, dürfte hieraus kein Lastproblem erwachsen. So sollte das System Anfragen von vielen 100 Studenten verarbeiten können, wenn man davon ausgeht, dass die Antworten nicht im Sekundentakt gegeben werden.

8.2 Praktischer Einsatz des Sytems

Der Prüfungs-Client konnte schon einmal während einer Probeprüfung im Mai Einsatz finden. Dabei wurden pro Durchlauf etwa 30 Kandidaten geprüft. Als Client-Betriebssystem wurde Windows verwandt, weil es das automatische Laden der Zertifikate von dem Memory-Stick vereinfacht und die User-Accounts bereits vom alten System her vorhanden waren. Da es bei diesem Test auch mehr um die Durchführbarkeit der Zertifikatausgabe ging, wurde auf eine Lock-Down-Umgebung verzichtet, so dass während der Tests sämtliche Funktionen des PC Arbeitsplatzes zur Verfügung standen.

8.2.1 Prüfungsraum

Zusammen mit der ISG-Informatik [1] wurden verschiedene Optionen der Durchführung für einen späteren Einsatz bei benoteten Leistungskontrollen besprochen. So könnte die ISG beispielsweise ein Netzwerk-Boot-Image in Form eines Linux-Systems verteilen, so dass die PCs im Prüfungsraum alle davon booten könnten. Das hätte den Vorteil, dass man so ein wirklich minimales System anbieten könnte ohne Webbrowser und mit Sperren, so dass die Prüfungssoftware nicht verlassen werden kann.
Vom Aufbau eines reinen Assessment-Raumes kann ich abraten, da die PCs dann nur während den Prüfungen genutzt werden könnten. Somit würde wertvolle Arbeitsfläche und PC-Hardware verschwendet, da diese die meiste Zeit ungenutzt wären.

8.2.2 Server

Nach mehreren von mir erbrachten Vorschlägen hat sich die Gruppe entschieden, einen Server der Firma SUN-Microsystems zu kaufen. Es handelt sich dabei um ein System der X2200 Serie, weist einen Arbeitsspeicher von 8GB auf, hat 4 CPU Kerne und 2 redundante 250GB Festplatten. Der Server ist wie in der Sicherheitsanalyse gefordert in einem 19“-Schrank in einem der ETH-Serverräume untergebracht. Als Software kommt Debian-Linux in der Version 4.0 „Etch“ zum Einsatz.

[1] `http://www.isg.inf.ethz.ch`

Literaturverzeichnis

[1] Siva Visveswaran, Dive into connection pooling with J2EE, 2000, [Online] http://java.sun.com/developer/technicalArticles/J2EE/pooling/

[2] The protection of information in computer systems, Saltzer, J.H. Schroeder, M.D., 1975 [Online] web.mit.edu/Saltzer/www/publications/protection/

[3] ELEKTRONISCHE UNTERSCHRIFT: RECHTLICHE BEDEUTUNG, GESETZLICHE REGELUNG, Susanne Keller, 4. April 1997, [Online] www.ifi.uzh.ch/ikm/Vorlesungen/inf_recht/1997/ZusDigSig.pdf

[4] Spezifikationen des E.Assessment System V1.0, Markus Dahinden, ETHZ [intern]

[5] Academic honesty and electronic assessment: tools to prevent students from cheating online Jose L. Cordova Paula Thornhill University of Louisiana at Monroe, Monroe, LA, Journal of Computing Sciences in Colleges archive Volume 22 , Issue 5 (May 2007)

[6] 26°C in EDV-Räumen – eine Temperatur ohne Risiko, Adrian Altenburger, Bundesamt für Energie, [online] http://www.bfe.admin.ch/

[7] Scripte der Vorlesung Client/Server Systeme WS 07/08, Prof. Dr. Wilhelm G. Spruth, Universität Tübingen

[8] EJB 3 professionell, Oliver Ihns, Dierk Harbeck, Stefan M. Heldt, Holger Koschek, ISBN: 3898644316

[9] Mastering Enterprise Java Beans 3.0 Rima Patel Sriganesh, Gerald Brose, Micah Silverman, ISBN 0471785415

[10] New Directions in Cryptography, Whitfield Diffie, Martin E. Hellman, 1976

[11] The Transport Layer Security (TLS) Protocol [Online] http://www.ietf.org/rfc/rfc4346.txt