DEPARTMENT FÜR INFORMATIK
SYSTEMSOFTWARE UND VERTEILTE SYSTEME

Implementierung einer Notenverwaltung

Individuelles Projekt

7. Mai 2009

Marc Hansen
Jann-Berghaus-Strasse 14
26409 Wittmund

Erstprüfer Prof. Dr.-Ing. Oliver Theel
Zweitprüfer Dipl.-Inform. Timo Warns

Erklärung zur Urheberschaft

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Oldenburg, den 7. Mai 2009

Marc Hansen

Bei der vorliegenden Ausarbeitung handelt es sich um ein individuelles Projekt der Abteilung Systemsoftware und verteilte Systeme der Universität Oldenburg. Sie wurde mit der Aufgabenstellung erstellt, den Notengebungsprozess dieser Abteilung zu vereinheitlichen und eine Webanwendung zu erstellen, welche den Prozess unterstützt. Diese Arbeit beschäftigt sich mit dem Prozess des Softwareengineerings für die Erstellung dieser Anwendung. So wird zunächst eine Anforderungsanalyse über das zu erstellende System durchgeführt um im Folgenden einen vollständigen Systementwurf durchzuführen. Dabei ist eine Webanwendung basierend auf einer PostgreSQL Datenbank und einem Apache Tomcat Webserver entstanden, welche mit geschützen Daten sicher umgehen kann und den Notengebungsprozess der Abteilung Systemsoftware und verteilte Systeme vereinheitlicht.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

In der Abteilung Systemsoftware und verteilte Systeme werden regelmäßig verschiedene Lehrveranstaltungen angeboten. Neben Seminaren und Praktika gibt es Vorlesungen mit oder ohne zusätzlichen Übungen, an deren Ende oft eine schriftliche Klausur oder eine mündliche Prüfung steht. Je nach Größe der Veranstaltungen werden ein oder mehrere Übungen angeboten, die durchaus auch von unterschiedlichen Tutoren betreut werden. In der Regel sind die Übungsaufgaben verpflichtend zu bearbeiten und werden durch die Tutoren benotet. Auf diese Weise fallen für eine Veranstaltung sehr viele Individual- und Gruppennoten an, aus denen am Ende eine Abschlussnote berechnet werden muss. Diese Berechnung erfolgt bisher mit Hilfe verschiedener Text-, CSV-, XML-Dateien, XSLT-Stylesheets, SQL-Skripten und Spreadsheets, was sehr umständlich, zeitraubend und fehleranfällig ist. Insbesondere ist die Verarbeitungskette nicht durchgängig, so dass immer wieder Dateien von Hand konvertiert werden müssen, bis die Daten schließlich in Stud.IP vorliegen. In dieser Arbeit soll eine Anwendung entwickelt werden, welche die erwähnten Vorgänge vereinfacht oder ersetzt, sodass ein einheitliches System für die Notenverwaltung entsteht, welches diesen Prozess sicherer und schneller macht. Hierzu wird ein Webbasiertes System entwickelt, welches auf einer Datenbank und einem Webserver basiert. Dies bietet den Vorteil, dass es von jedem Computer mit Internetzugang aus zugreifbar ist. Es soll den gesamten Prozess implementieren, sodass alle Text-, CSV-, XML-Dateien, XSLT-Stylesheets, SQL-Skripten und Spreadsheets überflüssig werden.
Die Ausarbeitung stellt den Prozess des Sofwareengineering nach dem Wasserfallmodell dar und gliedert sich in zwei große Teile. Teil 1 beschäftigt sich mit der Anforderungsanalyse des Projektes. Kapitel 2 gibt hierfür zunächst einen Überblick über das bestehende und das zu entwickelnde System. In Folge dessen werden in Kapitel 3 die Akteure des Systems vorgestellt. Kapitel 4 behandelt die funktionalen und nichtfunktionalen Anforderungen des Systems.
Teil 2 beschäftigt sich mit dem Entwurf des Systems. In Kapitel 5 wird der Entwurf eingeleitet und erste Entwurfsrichtlinien, die für das Vorgehen beim Entwurf und der Implementierung von Bedeutung sind, werden erfasst. Kapitel 6 beschäftigt sich mit dem Grobentwurf und beschreibt die Architektur des zu entwickelnden Systems. Der Feinentwurf findet in Kapitel 7 statt, welches die Software, seine Funktionalität und den Entwurf des Datenbankschemas beschreibt. In Kapitel 8 werden schließlich die Ergebnisse zusammengefasst, Alternativen zum entwickelten System aufgezeigt und ein Ausblick gegeben.

Teil I.

Anforderungsanalyse

2. Systembeschreibung

Der Verwaltungsaufwand der Notengebung in der Abteilung Systemsoftware und verteilte Systeme ist zur Zeit sehr hoch und fehleranfällig. Momentan werden die Noten in einer PostgreSQL Datenbank gespeichert. Ein Apache Cocoon Server hat Zugriff auf diese Noten, damit Studenten ihre Noten nach Eingabe der Matrikelnummer auslesen können. Das Speisen der Datenbank geschieht zur Zeit allerdings nicht einheitlich. So haben die Verantwortlichen eigene Methoden die Noten in die Datenbank zu schreiben. Das Verwalten der Noten geschieht z.B. durch Exceltabellen und XML. Zur Notenveröffentlichung wird dann ein eigens geschriebenes Skript ausgeführt, welches diese Tabellen in ein Tex-Dokument umformt, sodass es dann zu einer PDF-Datei kompilliert werden kann und schließlich am schwarzen Brett ausgehängt wird. Eine andere Methode ist ein SQL Skript, welches die Noten automatisch in die Datenbank schreibt. Um dieses System zu vereinheitlichen, bietet sich das Erstellen eines webbasiertes System an. Hierbei soll gezielt auf die Bedürfnisse einer Hochschulveranstaltung eingegangen werden, sodass das System nicht nur die Notenverwaltung beinhaltet, sondern die Verwaltung der Veranstaltungen im Ganzen. Dazu gehört auch die Verwaltung von Tutorien, Übungen, Prüfungen, Gruppen und anderen Dingen. Durch das Internet ist dieses System jederzeit und von überall zugreifbar und stets aktuell, sowohl in der Version des Datensatzes als auch der Software selbst. Da mit personenbezogenen Daten gearbeitet wird, spielt die Sicherheit des Systems eine wichtige Rolle. Grundlage des Systems soll das bestehende Websystem der Abteilung Systemsoftware und verteilte Systeme bestehend aus einem Apache Cocoon Server und einer PostgreSQL Datenbank sein.

3. Überblick und Beziehungen der Akteure

Da ein Notenverwaltungssystem von verschiedenen Personen mit unterschiedlichen Berechtigungen bedient wird, muss eine genaue Einteilung der Rechte stattfinden. Diese unterschiedlichen Berechtigungen werden im Folgenden anhand von Akteuren beschrieben.

- **Gast**
  Jeder Benutzer der sich unangemeldet im System befindet ist ein Gast . Dieser Akteur hat nur einen sehr eingeschränkten Zugang zum System. Es lassen sich lediglich veröffentlichte Notenverteilungen und allgemeine Informationen über die eingetragenen Veranstaltungen anzeigen.

- **Student**
  Ein Student besitzt keine Änderungsrechte an einer Veranstaltung. Er hat lediglich Zugriff auf Formulare in denen er Wünsche (z.B. Tutoriumszuordnung) angeben kann. Zusätzlich kann er eigene veröffentlichte Noten einsehen.

- **Tutor**
  Ein Tutor besitzt nur administrative Rechte für sein Tutorium. Er kann aber durch den Dozenten weitere Rechte erhalten, wenn dies notwendig ist. Sobald eine Veranstaltung als abgeschlossen gilt, verliert der Tutor alle seine Rechte für diese Veranstaltung.

- **Dozent**
  Ein Dozent ist ein Administrator mindestens einer Veranstaltung. Er besitzt alle Rechte für seine Veranstaltungen und kann diese an Tutoren ganz oder teilweise weitergeben.

- **Administrator**
  Ein Administrator ist der Systemverwalter. Er besitzt alle Rechte für das gesamte System und kann diese an andere weitergeben. Somit ist er auch dafür zuständig Dozenten zu ernennen.

Die betrachteten Akteure sind in einer Hierarchie eingeordnet in der der übergeordnete Akteur alle funktionalen Möglichkeiten des untergeordneten Akteurs erbt:

1. Administrator
2. Dozent
3. Tutor
4. Student
5. Gast

4. Funktionale und nicht-funktionale Anforderungen

Durch das Hierarchiesystem aus dem vorherigen Abschnitt ist es nun im Folgenden möglich für jeden Akteur Anwendungsfälle zu definieren, wobei jegliche Anwendungsfälle des untergeordneten Akteurs in der Hierarchie von den übergeordneten Akteuren übernommen werden. Somit wird mit den Anwendungsfällen des untersten Akteurs der Hierarchie begonnen. Hierbei wird jeder Anwendungsfall durch ein leicht angepassten Schema aus [Som07, S.112] beschrieben.

4.1. Gast

Ein Besucher des Systems, welcher dem System noch nicht bekannt ist, ist ein Gast. Er hat sehr eingeschränkte Möglichkeiten im System. Er kann lediglich alle öffentlichen Daten der Veranstaltungen einsehen. Außerdem kann er sich dem System durch Anmelden oder Registrierung bekannt machen. Er kann sich ein neues Passwort per E-Mail zuschicken lassen, wenn er bereits ein Account im System besitzt und sein Passwort vergessen hat. In Abbildung 4.1 sieht man einen Überblick der Anwendungsfälle.

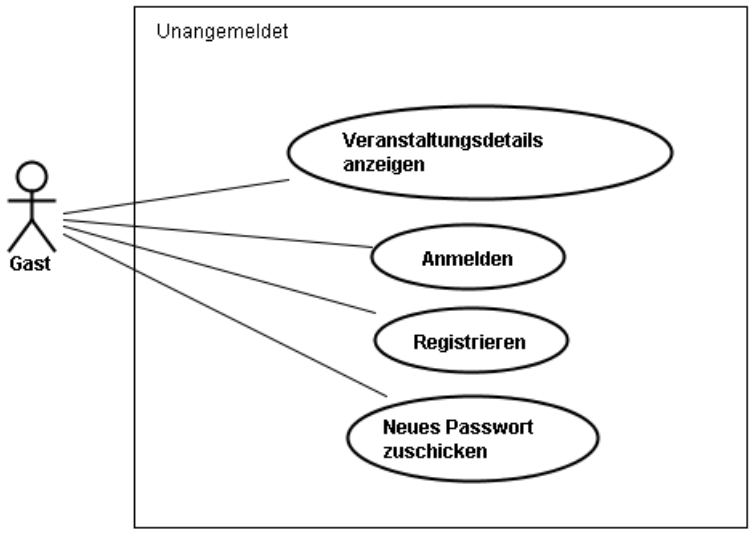

Abbildung 4.1.: Anwendungsfalldiagramm „Unangemeldet - Gast“

4.1.1. Veranstaltungsdetails anzeigen (GA-01)

In diesem Anwendungsfall, kann der Gast sich Informationen zu den im System vorhandenen Veranstaltungen anzeigen lassen. Sowohl aktuelle als auch abgeschlossene Veranstaltungen können eingesehen werden. Es sind unterschiedliche Informationen wie Termine, Dozenten, Teilnehmer, Raumpläne, Art der Veran-

staltung, Anzahl der Kreditpunkte, usw. denkbar. Angezeigt werden allerdings nur Daten die nicht vom Datenschutz betroffen sind. Hier muss vorher eine klare Einteilung im Entwurf vorgenommen werden.

Anwendungsfall	GA-01
Name	Veranstaltungsdetails anzeigen
Initiierender Akteur	Gast
Weitere Akteure	-
Kurzbeschreibung	Es werden jedem Gast eine Liste von Veranstaltungen angezeigt in der weitere öffentliche Informationen über diese erhältlich sind.
Vorbedingung	Gast betritt System
Nachbedingung	Gast erhält erwünschte Informationen
Ablauf	Der Gast wählt eine Option zum Anzeigen einer Liste von aktuellen Veranstaltungen, kann allerdings auch Veranstaltungsinformationen über ältere Veranstaltungen einsehen. Hierzu gibt es entsprechende Zugangsoptionen.
Extends und Includes	keine

4.1.2. Anmelden (GA-02)

Der Gast kann sich dem System bekannt machen indem er sich mit seinen Zugangsdaten zu einem vorher registrierten Account anmeldet. Hierdurch erhält er weitere Rechte verschiedene Funktionen im System zu verwenden. Die Zugangsdaten bestehen aus seiner E-Mailadresse und einem Passwort.

Anwendungsfall	GA-02
Name	Anmelden
Initiierender Akteur	Gast
Weitere Akteure	-
Kurzbeschreibung	Ein Gast kann sich anmelden.
Vorbedingung	Gast betritt System
Nachbedingung	Gast ist angemeldet und wird zum Administrator, Dozent, Tutor oder Student und erhält die Möglichkeit entsprechende Funktionen des Systems zu verwenden.
Ablauf	Der Gast muss eine E-Mailadresse und ein Passwort eingeben.
Extends und Includes	keine

4.1.3. Registrieren (GA-03)

Der Gast hat die Möglichkeit sich dem System durch eine Registrierung bekannt zu machen. Hierfür muss er mindestens eine E-Mailadresse und seinen vollen Namen angeben. Die E-Mailadresse darf nicht bereits durch einen anderen Account im System verwendet werden. Es wird dann eine E-Mail mit einem Link an die angegebene Adresse geschickt mit dessen Hilfe der Gast seinen Account aktivieren kann. Ein neu-

er Account gilt als Student-Account bis dieser durch einen Dozent oder Administrator in der Hierarchie hochgestuft wird.

Anwendungsfall	GA-03
Name	Registrieren
Initiierender Akteur	Gast
Weitere Akteure	-
Kurzbeschreibung	Der Gast kann sich Registrieren.
Vorbedingung	Gast gibt eine gültige und nicht vom System verwendete E-Mailadresse und seinen vollständigen Namen an
Nachbedingung	Ein neuer Account wurde eingerichtet und der Gast erhält eine Bestätigung per E-Mail in der er über einen Link seinen Account aktivieren kann.
Ablauf	Der Gast erhält ein Formular in dem er diverse Daten eingeben kann. Hierzu gehören mindestens eine E-Mailadresse und sein vollständiger Name. Es ist möglich weitere optionale Informationen anzugeben.
Extends und Includes	keine

4.1.4. Neues Passwort zuschicken (GA-04)

Der Gast kann sich ein neues Passwort zuschicken lassen, falls er seines vergessen hat. Hierfür gibt er seine E-Mailadresse ein, und das System generiert für den Account dieser E-Mailadresse ein neues Passwort und sendet es. Das alte Passwort wird verworfen.

Anwendungsfall	GA-04
Name	Neues Passwort zuschicken
Initiierender Akteur	Gast
Weitere Akteure	-
Kurzbeschreibung	Der Gast kann mithilfe seiner E-Mailadresse ein neues Passwort anfordern.
Vorbedingung	Gast gibt eine gültige E-Mailadresse ein
Nachbedingung	Das Passwort des Accounts wurde geändert und der Gast erhält das neue Passwort per E-Mail zu gesendet.
Ablauf	Der Gast gibt seine E-Mailadresse ein und fordert ein neues Passwort an.
Extends und Includes	keine

4.2. Student

Im Folgenden werden die Möglichkeiten des Akteurs Student und damit jedes angemeldeten Besuchers beschrieben. Er kann seine persönlichen Daten ändern, zu denen sein Name, Passwort und eine Anschrift gehören. Außerdem besitzt jeder angemeldete Benutzer die Möglichkeit einige Systemoptionen für sich einzustellen. Vorgesehen hierfür ist mindestens das automatische Senden von Benachrichtigen ein- und

auszuschalten. Weitere Möglichkeiten wären Themeneinstellungen der Darstellung des Systems sowie Startbildschirme zu erstellen in denen nur gewünschte Informationen zu sehen sind. Diese beiden Features sind aber nicht zwingend erforderlich und sollen daher nur implementiert werden, wenn die Zeit ausreicht. Der Student kann sich zu Veranstaltungen an und abmelden. Dies gilt nur für aktive Veranstaltungen. Außerdem kann er seine eigenen Noten einsehen soweit diese freigegeben wurden. In Abbildung 4.2 sieht man einen Überblick über die Anwendungsfälle des Studenten.

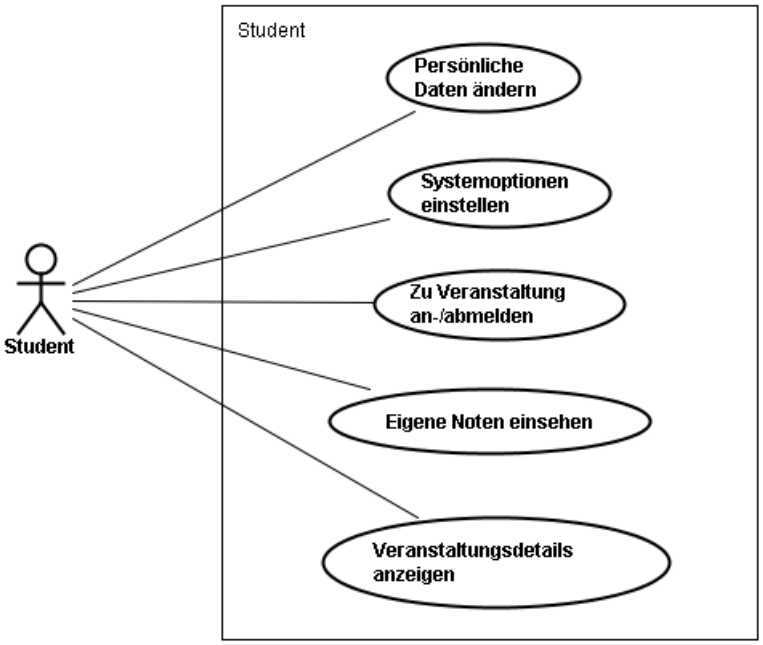

Abbildung 4.2.: Anwendungsfalldiagramm „Angemeldet - Student“

4.2.1. Persönliche Daten ändern (ST-01)

Der Student hat die Möglichkeit seine persönlichen Daten zu ändern. Hierzu gehören sein Passwort, sein Titel, seine Anschrift, seine Telefonnummer und eine Homepage. Seinen Namen und seine E-Mailadresse kann er nach der Anmeldung nicht mehr ändern.

Anwendungsfall	ST-01
Name	Persönliche Daten ändern
Initiierender Akteur	Student
Weitere Akteure	-
Kurzbeschreibung	Der Student hat die Möglichkeit seine persönlichen Daten zu ändern.
Vorbedingung	Der Student muss hierfür angemeldet sein
Nachbedingung	Der Student hat seine Daten geändert.
Ablauf	Der Student erhält ein Formular in dem er entsprechende Änderungen vornehmen kann.
Extends und Includes	keine

4.2.2. Benachrichtigungsoptionen einstellen (ST-02)

Der angemeldete Benutzer kann seine persönlichen Benachrichtigungsoptionen einstellen. Diese Einstellungen wirken sich auf die E-Mailverteilung von Benachrichtigungen aus. Benachrichtigungen sind für folgende Systemereignisse ein und ausschaltbar:

- Änderungen der Veranstaltungsdetails einer eigenen Veranstaltung.
- Änderungen bei den eigenen Noten.
- Tutoriumszuordnungen und Gruppenzuordnungen die nicht selbst getätigt wurden.
- Änderungen der eigenen Rechte.

Anwendungsfall	ST-02
Name	Benachrichtigungsoptionen einstellen
Initiierender Akteur	Student
Weitere Akteure	-
Kurzbeschreibung	Der eingeloggte Benutzer hat die Möglichkeit festzulegen welche Typen von Benachrichtigungen er per E-Mail erhalten will.
Vorbedingung	Der Benutzer ist eingeloggt.
Nachbedingung	Das Benachrichtigungssystem hat die Änderungen übernommen.
Ablauf	Der Benutzer erhält ein Formular in dem er einzelne Optionen an und ausschalten kann.
Extends und Includes	

4.2.3. Zu Veranstaltung an-/abmelden (ST-03)

Der Benutzer kann sich zu aktiven Veranstaltungen anmelden. Das sind Veranstaltungen die noch nicht abgeschlossen sind. Diese Veranstaltungen sollen in der Liste zu erkennen sein.

Anwendungsfall	ST-03
Name	Zu Veranstaltung an-/abmelden
Initiierender Akteur	Student
Weitere Akteure	-
Kurzbeschreibung	Der Student kann sich zu Veranstaltungen anmelden.
Vorbedingung	Der Student ist eingeloggt. Die Veranstaltung ist aktiv.
Nachbedingung	Der Student hat sich angemeldet und wurde zu den Veranstaltungsteilnehmern hinzugefügt.
Ablauf	Der Student wählt eine Veranstaltung aus einer Liste aus und benutzt die entsprechende Option zum Anmelden.
Extends und Includes	

4.2.4. Eigene Noten einsehen (ST-04)

Der Student kann seine eigenen Noten einsehen. Er erhält eine Gesamtübersicht über seine Noten und alle freigegebenen Zwischennoten. Seine Noten werden auch angezeigt wenn er sich Notenübersichten von Veranstaltungen anzeigen lässt in denen er selbst eine Note erhalten hat.

Anwendungsfall	ST-04
Name	Eigene Noten einsehen
Initiierender Akteur	Student
Weitere Akteure	-
Kurzbeschreibung	Der Student kann seine eigenen Noten einsehen.
Vorbedingung	Der Student ist eingeloggt.
Nachbedingung	Der Student erhält die gewünschten Informationen.
Ablauf	Der Student klickt die entsprechende Option im Menü an.
Extends und Includes	

4.3. Tutor

Der Tutor hat wie in Abschnitt 3 beschrieben, alle funktionalen Möglichkeiten wie der untergeordnete Student. Der Tutor besitzt außerdem ein oder mehrere Tutorien, welche er verwaltet. Er erhält hierdurch auch Einblick in die Noten der Studenten die in seinem Tutorium sind. Alle anderen Noten bleiben ihm verborgen. Abbildung 4.3 gibt einen Überblick über die funktionalen Möglichkeiten, die dem Tutor für die Verwaltung seiner Tutorien zur Verfügung stehen.

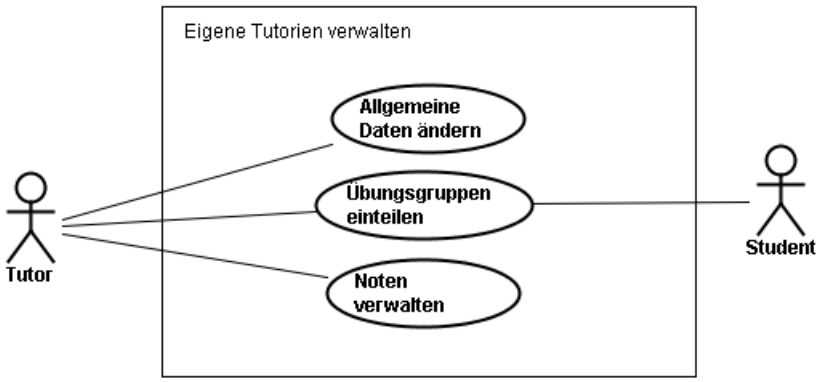

Abbildung 4.3.: Anwendungsfalldiagramm „Angemeldet - Tutor“

4.3.1. Allgemeine Daten ändern (TU-01)

Jedes Tutorium besitzt eine Reihe allgemeiner Daten, die vom Tutor verändert werden können. Hierzu gehören eine Angabe des Ortes, der Zeit, der Dauer und der maximalen Anzahl an Teilnehmern.

Anwendungsfall	TU-01
Name	Allgemeine Daten ändern
Initiierender Akteur	Tutor
Weitere Akteure	-
Kurzbeschreibung	Der Tutor hat die Möglichkeit allgemeine Daten des Tutoriums zu ändern.
Vorbedingung	Der Tutor benötigt die Rechte für das jeweilige Tutorium.
Nachbedingung	Allgemeine Daten wurden geändert.
Ablauf	Der Tutor erhält ein Formular mit den aktuellen Einstellungen die er dort in geeigneter Weise ändern kann.
Extends und Includes	keine

4.3.2. Übungsgruppen einteilen (TU-02)

Der Tutor kann in seinem Tutorium angemeldete Studenten zu Übungsgruppen zusammenfassen. Dabei steht ihm grundsätzlich die Möglichkeit zur Verfügung, festzulegen wie groß diese Gruppen sein sollen. Zusätzlich kann er verschiedene Verfahren der Zuordnung wählen: Zufällige Zuordnung, Zuordnung nach Wahl der Studenten(Erstwunsch, Zweitwunsch, Drittwunsch..) oder eigene Einteilung.

Anwendungsfall	TU-02
Name	Übungsgruppen einteilen
Initiierender Akteur	Tutor
Weitere Akteure	Student
Kurzbeschreibung	Der Tutor kann Übungsgruppen einteilen.
Vorbedingung	Der Tutor besitzt die Rechte für das jeweilige Tutorium.
Nachbedingung	Die Einteilung wurde vorgenommen.
Ablauf	Der Tutor wählt die Art der Zuordnung aus. Die Studenten erhalten eine Benachrichtigung, dass sie sich für Übungsgruppen einteilen können, oder zugeteilt wurden. Im ersten Fall geben die Studenten ihre Wunschgruppen an. Der Tutor muss einen Endzeitpunkt bestimmen, an dem die automatische Zuordnung nach den vorhandenen Daten erfolgt.
Extends und Includes	keine

4.3.3. Noten verwalten (TU-03)

Der Tutor kann Zwischennoten für Studenten seines Tutoriums eintragen und ändern. Die geänderten Noten werden visuell markiert um die Veränderung anzuzeigen. Der Dozent kann diese bestätigen um die Markierung zu entfernen. Auf die Errechnung der Endnoten hat der Tutor keinen Einfluss. Diese wird in Abschnitt 4.4 betrachtet.

Anwendungsfall	TU-03
Name	Noten verwalten
Initiierender Akteur	Tutor
Weitere Akteure	-
Kurzbeschreibung	Der Tutor kann Zwischennoten für seine Studenten verwalten.
Vorbedingung	Der Tutor besitzt die Rechte für das jeweilige Tutorium.
Nachbedingung	Die Noten wurden verändert, diese werden bis zur Bestätigung durch den Dozent markiert um die Veränderung anzuzeigen.
Ablauf	Der Tutor erhält eine Liste aller Studenten seines Tutoriums. Hier sieht er alle Zwischennoten und kann diese direkt verändern.
Extends und Includes	keine

4.4. Dozent

Der Dozent besitzt alle Rechte an seinen eigenen Veranstaltungen. Er kann neue Veranstaltungen erstellen und diese verwalten. Er hat auch die Möglichkeit einen Studenten zum Tutor zu ernennen indem er ihm ein Tutorium zuteilt. Er kann außerdem einem Tutor alle oder Teile seiner Rechte geben, damit dieser die Verwaltung der Veranstaltung im System übernehmen kann. In Abbildung 4.4 sieht man einen Überblick über die betrachteten Anwendungsfälle des Dozenten.

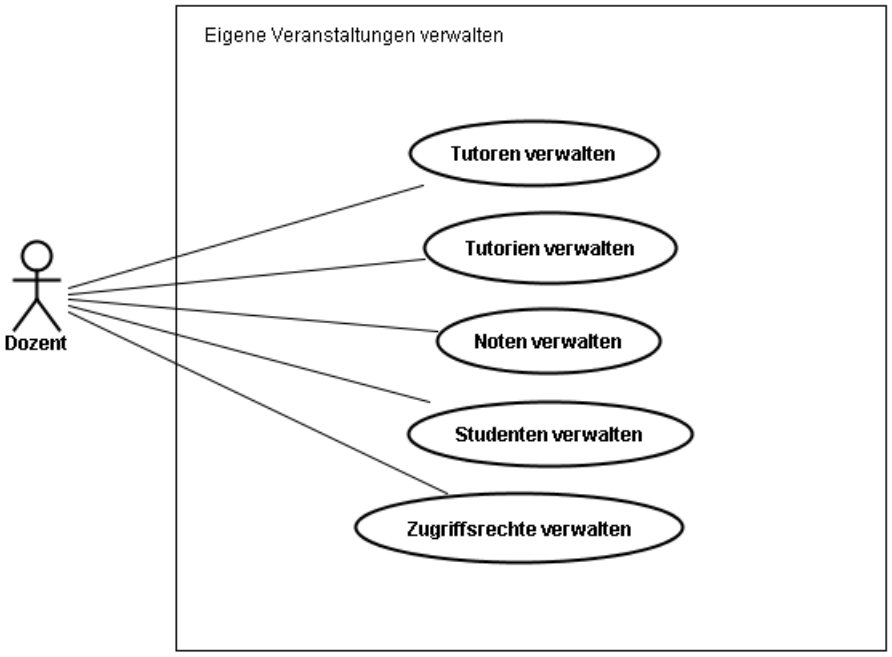

Abbildung 4.4.: Anwendungsfalldiagramm „Angemeldet - Dozent“

4.4.1. Tutoren verwalten (DO-01)

Der Dozent kann einen Studenten zum Tutor ernennen und den Tutor-Status wieder entfernen indem er ihm ein Tutorium zuteilt oder wieder entfernt.

Anwendungsfall	DO-01
Name	Tutoren verwalten
Initiierender Akteur	Dozent
Weitere Akteure	-
Kurzbeschreibung	Der Dozent kann Tutoren ernennen.
Vorbedingung	Der Dozent benötigt die Rechte über eine Veranstaltung und muss bereits ein Tutorium eingerichtet haben.
Nachbedingung	Der Student wird zum Tutor und wurde dem Tutorium zugeordnet. Er erhält die entsprechenden Rechte über das Tutorium.
Ablauf	Der Dozent ordnet einem vorher eingerichteten Tutorium einen Tutor zu, indem er diesen aus der Liste aller zur Veranstaltung angemeldeten Studenten auswählt.
Extends und Includes	keine

4.4.2. Tutorien verwalten (DO-02)

Der Dozent kann für seine Veranstaltung Tutorien erstellen. Diese sind nicht aktiv und damit auch nicht sichtbar für andere, solange kein Tutor zu diesem Tutorium zugeordnet wurde. Eine Zuordnung von Studenten zu einem Tutorium kann entweder zufällig, durch ein Wahlsystem oder durch eigene Zuordnung geschehen.

Anwendungsfall	DO-02
Name	Tutorien verwalten
Initiierender Akteur	Dozent
Weitere Akteure	-
Kurzbeschreibung	Der Dozent kann Tutorien erstellen und verwalten.
Vorbedingung	Der Dozent benötigt die Rechte über eine Veranstaltung.
Nachbedingung	Ein Tutorium wurde erstellt oder es wurden gewünschte Änderungen vorgenommen.
Ablauf	Der Dozent erstellt ein Tutorium oder trägt allgemeine Daten ein. Die Zuordnung vom Studenten kann durch eine Liste von Studierenden geschehen.
Extends und Includes	keine

4.4.3. Noten verwalten (DO-03)

Der Dozent kann die Noten seiner Veranstaltungen verwalten. Er kann Zwischennoten anlegen und die Endnotenberechnung einstellen. Diese Berechnung kann jederzeit verändert werden, was eine dynamische Anpassung der Noten an die Durchschnittsleistung der Studenten möglich macht.

Anwendungsfall	DO-03
Name	Noten verwalten
Initiierender Akteur	Dozent
Weitere Akteure	-
Kurzbeschreibung	Der Dozent kann Noten verwalten.
Vorbedingung	Der Dozent benötigt die Rechte über eine Veranstaltung.
Nachbedingung	Die gewünschten Änderungen am Notensystem wurden vorgenommen.
Ablauf	Der Dozent erhält eine Liste aller Zwischennoten und kann diese direkt ändern oder neue anlegen. Durch eine geeignete Benutzungsoberfläche kann er die Endnotenberechnung modifizieren.
Extends und Includes	keine

4.4.4. Studenten verwalten (DO-04)

Der Dozent kann seine Studenten verwalten. Er kann Studenten aus seiner Veranstaltung entfernen, oder sie in Tutorien zuordnen. Außerdem kann er Zwischennoten eintragen oder auch Endnoten anpassen, auch wenn diese dynamisch generiert wurden. Er hat somit volle Kontrolle über die Noten der Studenten.

Anwendungsfall	DO-04
Name	Studenten verwalten
Initiierender Akteur	Dozent
Weitere Akteure	-
Kurzbeschreibung	Der Dozent kann Studenten verwalten.
Vorbedingung	Der Dozent benötigt die Rechte über eine Veranstaltung.
Nachbedingung	Die gewünschten Änderungen wurden vorgenommen.
Ablauf	Der Dozent sieht eine übersichtliche Liste seiner Studenten und deren eingetragener Noten und kann die gewünschten Änderungen direkt in dieser Liste vornehmen.
Extends und Includes	keine

4.4.5. Zugriffsrechte verwalten (DO-05)

Der Dozent kann die Rechte für seine Veranstaltungen teilweise oder vollständig an Tutoren weitergeben. Somit ist es möglich, dass seine Veranstaltungen vollständig durch die jeweiligen Tutoren im System verwaltet werden.

Anwendungsfall	DO-05
Name	Zugriffsrechte verwalten
Initiierender Akteur	Dozent
Weitere Akteure	-
Kurzbeschreibung	Der Dozent kann die Rechte seiner Tutoren verwalten.
Vorbedingung	Der Dozent besitzt die Veranstaltung.
Nachbedingung	Die gewünschten Änderungen an der Rechteverteilung wurden vorgenommen.
Ablauf	Der Dozent erhält eine Liste seiner Tutoren und kann eine Reihe von Rechten durch anklicken bei einzelnen Tutoren aktivieren oder deaktivieren.
Extends und Includes	keine

4.5. Administrator

Der Administrator ist der Superuser des Systems. Er besitzt alle Rechte der anderen Akteure und kann als einziger Akteur Dozenten ernennen und löschen. Außerdem kann er Benutzeraccounts und Veranstaltungen löschen. Abbildung 4.5 gibt einen Überblick über die betrachteten Anwendungsfälle des Administrators.

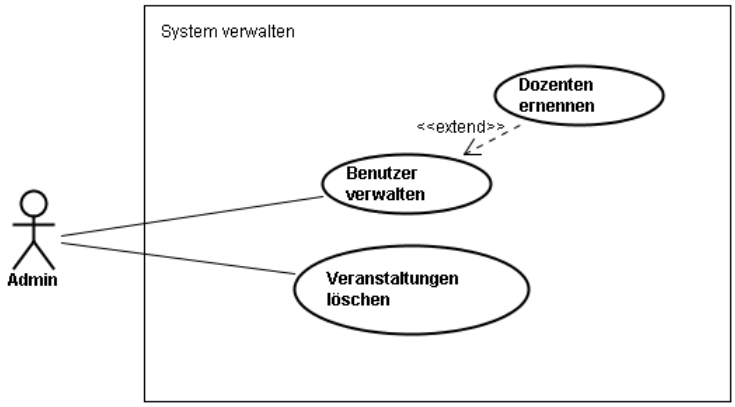

Abbildung 4.5.: Anwendungsfalldiagramm „System verwalten“

4.5.1. Benutzer verwalten (AD-01)

Der Administrator kann Benutzer verwalten. Hierzu gehört das Ändern jeglicher Daten und das Löschen von Benutzern.

Anwendungsfall	AD-01
Name	Benutzer verwalten
Initiierender Akteur	Administrator
Weitere Akteure	-
Kurzbeschreibung	Der Administrator kann Benutzer verwalten.
Vorbedingung	Der Benutzer ist als Administrator angemeldet.
Nachbedingung	Die gewünschten Änderungen wurden vorgenommen.
Ablauf	Der Administrator ändert eingetragene Daten durch das Eintragen in ein Formular, oder entfernt Benutzer durch eine Suche oder Auswahl aus einer Liste.
Extends und Includes	Dozenten ernennen

4.5.2. Dozenten ernennen (AD-02)

Der Administrator kann als einziger Akteur neue Dozenten ernennen oder den Dozent Status von Dozenten entfernen.

Anwendungsfall	AD-02
Name	Dozenten ernennen
Initiierender Akteur	Administrator
Weitere Akteure	-
Kurzbeschreibung	Der Administrator kann Benutzer zu Dozenten befördern.
Vorbedingung	Der Benutzer ist als Administrator angemeldet.
Nachbedingung	Der ausgewählte Benutzer ist jetzt Dozent und besitzt die entsprechenden Rechte.
Ablauf	Der Administrator wählt einen Benutzer aus einer Liste oder einer Suchfunktion aus den er zum Dozent ernennen will.
Extends und Includes	

4.5.3. Veranstaltungen löschen (AD-03)

Der Administrator kann als einziger Benutzer Veranstaltungen löschen. Hierdurch werden alle zugeordneten Noten und Tutorien gelöscht.

Anwendungsfall	AD-03
Name	Veranstaltungen löschen
Initiierender Akteur	Administrator
Weitere Akteure	-
Kurzbeschreibung	Der Administrator kann Veranstaltungen löschen.
Vorbedingung	Der Benutzer ist als Administrator angemeldet.
Nachbedingung	Die ausgewählte Veranstaltung wurde gelöscht.
Ablauf	Der Administrator wählt eine Veranstaltung aus einer Liste oder durch eine Suchfunktion aus und löscht sie durch Anklicken der jeweiligen Option. Der Vorgang muss bestätigt werden, weil alle enthaltenen Noten verloren gehen.
Extends und Includes	

4.6. Nicht-funktionale Anforderungen

Das System enthält diverse nicht-funktionale Anforderungen. Die Aufgabenstellung des individuellen Projektes bietet bereits einige dieser Anforderungen. So soll das System möglichst in das bestehende System der Abteilung für Systemsoftware und verteilte Systeme integriert werden. Dieses beinhaltet einen Apache-Cocoon HTTP-Server, welcher eine Servlet-Engine beinhaltet und sich daher für die Verwendung von Java Server Pages eignet. Außerdem ist eine PostgreSQL Datenbank vorhanden, die benutzt werden sollte. Eine zusätzliche Anforderung ist eine Offline-Version des Systems, welche ohne eine Internetverbindung auskommt. Beim Umgang mit personenbezogenen Daten ist auf sichere Verarbeitung und Kommunikation im System zu achten.

4.6.1. Benutzungsfreundlichkeit

Auf die Benutzungsfreundlichkeit des Systems muss besonders großen Wert gelegt werden. Die Implementierung des Systems soll vor allem den Vorgang der Notengebung beschleunigen und vereinfachen. Damit dies möglich ist, muss das System leicht verständlich und intuitiv bedienbar sein. Dies bedeutet, dass lange komplizierte Umwege über verschiedene Webseiten vermieden werden sollten. Der Benutzer soll die Möglichkeit haben, so schnell wie möglich seine gewünschte Funktion auszuführen. Außerdem soll der Zugriff auf Daten, sowie die Änderung dieser Daten einfach durch das Eintragen in Listen geschehen. Ein Beispiel hierfür wäre die Verwaltung der Noten: Der Dozent sollte eine Liste mit all seinen Studenten sehen in der er direkt die entsprechenden Noten eintragen kann. Des Weiteren sollte ein komplettes Nachladen der Webseite vermieden werden, wo es möglich ist. Dies verhindert, dass ein Benutzer sich nach dem Wechsel zu einer anderen Webseite neu orientieren muss.

4.6.2. Rechtesystem

Die einzelnen Aufgabenbereiche bei der Notengebung erfordern ein genaues Rechtesystem, dass die Zugriffsmöglichkeit einzelner Akteure beschränkt. Jeder Akteur darf nur auf die personenbezogenen Daten

Einsicht haben, die er für die Ausübung seiner Tätigkeiten benötigt. Daher zieht sich das Rechtesystem durch die komplette Anwendung und ist ein zentraler Bestandteil aller Anwendungsmöglichkeiten.

4.6.3. Persistente Datenhaltung

Das Projekt benötigt eine Möglichkeit um Daten persistent zu speichern. Als Teil der Aufgabenstellung steht hierfür eine PostgreSQL Datenbank zur Verfügung dessen Schema angepasst werden darf. Änderungen des Datenbankzustands, vor allem bei der Änderung von Noten, sollen möglichst exakt mitgeloggt werden.

4.6.4. Sichere Kommunikation

Da es um personenbezogene Daten geht, muss die Kommunikation des Systems besonders geschützt werden. Jeglicher Datenverkehr muss daher über eine verschlüsselte Verbindung stattfinden.

4.6.5. Offline-Tool

Eine zusätzliche Anforderung ist ein Subsystem, welches auch ohne Internetverbindung verwendbar ist. In diesem System soll es möglich sein Daten zu verändern und mit dem Hauptsystem zu synchronisieren sobald eine Internetverbindung bereit steht. Dies soll das Eintragen von Noten auch unterwegs angenehm und schnell machen. Dies ist eine optionale Anforderung und wird nur implementiert, wenn die Zeit ausreicht.

4.6.6. Erweiterbarkeit

Der Programmcode soll so modular und übersichtlich sein, dass ein späteres Reengineering einfach möglich ist um zum Beispiel neue Veranstaltungsformen unterstützen zu können. Dies soll durch eine genaue Dokumentation der Softwarekomponenten und durch Einhaltung von strengen Codekonventionen unterstützt werden.

Teil II.

Entwurf

5. Einleitung

Nachdem die Anforderungen an das System im vorherigen Kapitel erfasst wurden, wird im Folgenden ein Entwurf des beschriebenen Systems erstellt. Hierbei wird zunächst auf die Entwurfsziele eingegangen, in denen genauer spezifiziert wird, auf welche Eigenschaften der Software besonderen Wert gelegt werden soll. Danach wird auf die verwendete Architektur des Systems eingegangen und ein objektorientierter Entwurf des Datenmodells erstellt, welcher für den Entwurf der Benutzeroberfläche verwendet wird. Zum Schluss dieses Entwurfs wird ein Datenbankentwurf durchgeführt um Daten persistent speichern zu können.

5.1. Entwurfsrichtlinien

5.1.1. Codekonventionen

Programmiert und dokumentiert wird der Code in Englisch. Die Dokumentation des Codes wird durch JavaDoc erstellt. Es ist also darauf zu achten, dass die Kommentare an den Softwarekomponennten JavaDockonform sind. Des Weiteren soll der Javacode den Codekonventionen vom Sun Developer Network (SDN, siehe [SDNa]) entsprechen um diesen besser lesbar zu machen.

5.1.2. Datenbank

Das Datenbankschema soll möglichst redundanzfrei sein, damit die Datenbank nicht in einen inkonsistenten Zustand geraten kann. Wichtig hierfür ist es auch jegliche Fremdschlüsselbeziehungen zu identifizieren. Außerdem werden für schreibende Zugriffe auf die Datenbank stets Transaktionen verwendet, sodass der Datenbankzustand im Fehlerfall zurückgesetzt werden kann. Dies ist vor allem bei voneinander abhängigen Transaktionen sehr wichtig. Des Weiteren sollen jegliche Änderungen am Datenbankzustand in einer eigenen Relation mitgeloggt werden. Dies soll softwareseitig durch die Klasse DatabaseManager (Abschnitt 7.4.2.1) geschehen.

6. Grobentwurf

Die Systemarchitektur ist durch die Anforderungen schon stark eingeschränkt. So soll eine Webanwendung erstellt werden, die über einen Browser gesteuert wird und eine Datenbank zum persistenten Speichern von Daten verwendet. Diese Architektur bietet diverse Vorteile. Die Verwaltungssoftware ist über das Internet von überall aus erreichbar. Sie muss auf dem Client nicht installiert werden, sondern wird durch einen aktuellen Webbrowser gesteuert. Durch ein Websystem hat man direkten Zugriff auf eine zentrale Datenbank. Nachteil ist allerdings, dass die Verbindung zwischen Client und Server besonders geschützt werden muss, weil sonst Unbefugte Zugriff zu sensiblen Daten erhalten könnten. Die genauen Komponenten, sowie die Kommunikation zwischen den Komponenten dieser Architektur werden im Folgenden erläutert.

6.1. Komponenten

Die Systemarchitektur entspricht der typischen Client/Server Architektur einer Webanwendung, bestehend aus einem Webserver, einer Datenbank und einem Webbrowser, der die Benutzeroberfläche anzeigt. Abbildung 6.1 zeigt einen Überblick über das System.

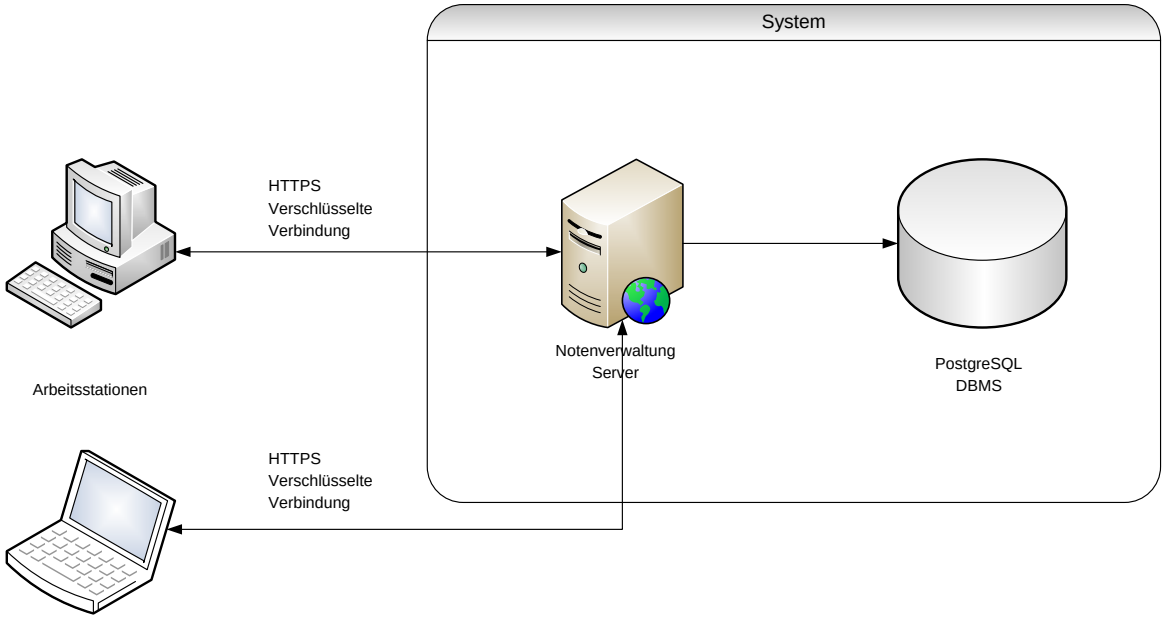

Abbildung 6.1.: Grobarchitektur

6.1.1. Client

Ein Webbrowser stellt den Client des Systems dar. Er bietet die Schnittstelle zum Benutzer des Systems. Diese Anforderung bringt jedoch einige Probleme mit sich. Der Markt hat eine ganze Reihe von Webbrowsern zum Vorschein gebracht. So gut wie jeder Browser hat seine Eigenheiten, sowohl was den Umfang des Befehlssatzes von Javascript und Cascading Style Sheets angeht, als auch die Umsetzung einzelner Befehle. Das Ziel muss es sein, das System für möglichst viele Benutzer zugänglich zu machen. Wie Abbildung 6.2 zeigt, sind der „Internet Explorer“ ab Version 6.0 und „Firefox“ ab Version 2.0 die zwei meist benutzen Browser für den PC in Deutschland. Für Mac Benutzer sollte „Safari“ unterstützt werden. Für den Client werden also HTML Seiten programmiert, welche durch Javascript dynamische Aspekte erhalten und mithilfe von Cascading Style Sheets formatiert sind. Aktuelle Javascript Frameworks unterstützen die erwähnten gängigen Browser und bieten somit eine gute Möglichkeit platformunabhängig zu programmieren. In Abschnitt 7.3.2 werden verschiedene gängige Frameworks untersucht und eines ausgewählt.

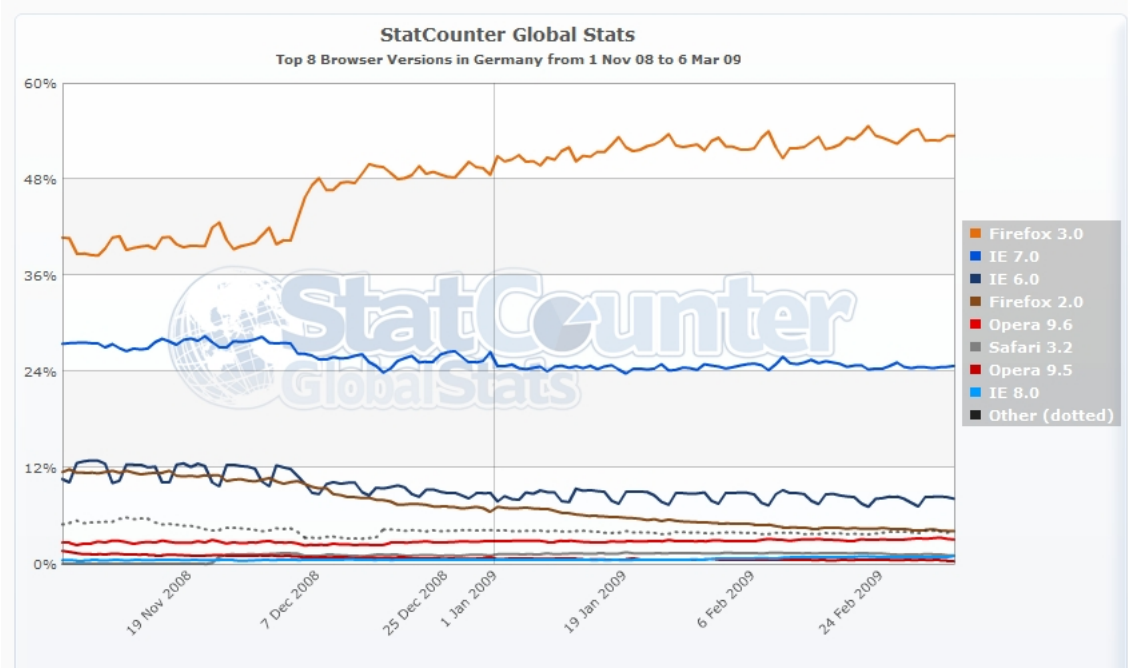

Abbildung 6.2.: Marktverteilung Browser (Quelle: `http://gs.statcounter.com` ,06.03.2009)

6.1.2. Server

Beim Server handelt es sich um einen Apache Tomcat 6.0.18, da für die dynamische Komponente des Systems Java 6 und Java Server Pages wegen guter Erfahrungen als Programmiersprache gewählt wurden. Mit Java ist es möglich einfach objektorientierten Code zu entwickeln. Des Weiteren ist die Sprache sehr gut dokumentiert, und wird gut durch kostenlose Entwicklungsumgebungen, wie Eclipse oder Netbeans unterstützt. Apache Tomcat 6.0.18 unterstützt außerdem bereits JSP 2.1, sodass die Java Server Pages Standard Tag Library (kurz JSTL, Abschnitt 7.5.4) ohne weiteres verwendet werden kann. Diese bietet diverse Möglichkeiten, Aufgaben einfacher und übersichtlicher zu programmieren.

6.1.3. Datenbank

Um Daten auch persistent speichern zu können, wird eine Datenbank verwendet. In diesem Fall wird die durch die Anforderungsanalyse angegebene PostgeSQL Datenbank (Version 8.3.6-1) der Abteilung Systemsoftware und verteilte Systeme verwendet.

6.2. Kommunikation

Da bei der Kommunikation persönliche Daten, wie Noten oder Matrikelnummern versendet und empfangen werden, muss die Verbindung besonders geschützt werden. Besonders die Client/Server Verbindung gilt es zu schützen, da diese über das Internet stattfindet.

6.2.1. Client-Server Kommunikation

Ein Datenpaket, das im Internet von einem Ort zu einem anderen geschickt wird, passiert meistens viele Zwischenstationen, wie z.B. Router, Gateways, Switches und Hubs. Da das HTTP (HyperText Transfer Protocol, [Gro99, RFC2616]), welches im Allgemeinen zur Übertragung von Daten im Internet verwendet wird, ein Klartext-Protokoll ist, kann jede der genannten Komponenten die Nachrichten lesen die versendet werden. Ein Problem sind vor allem die Hubs, da diese eine Nachricht an alle angeschlossenen Hosts weiter sendet. Die Netzwerkhardware dieser Hosts, merkt zwar dass die Datenpakete nicht an sie adressiert wurden und verwerfen diese normalerweise, allerdings kann ein potentieller Angreifer alle angekommenen Pakete mitschneiden. Dies wird auch als Sniffing bezeichnet und ermöglicht es z.B. Passwörter, Matrikelnummern und andere persönliche Daten zu lesen. Wird die Verbindung jedoch verschlüsselt, so kann ein solcher Angreifer nur mitlesen, wenn er weiß welcher Schlüssel benutzt wurde. In dem zu entwickelnden System wird also zwischen Client und Server (also zwischen Browser und Apache Tomcat) eine verschlüsselte Verbindung benutzt. Diese wird durch den Einsatz von HTTPS (HyperText Transfer Protocol Secure [Gro00, RFC2818]) erreicht, das ähnlich wie das HTTP funktioniert, allerdings SSL/TLS (siehe auch [Gro08, RFC5246]) als zusätzliche Schicht zwischen TCP und HTTP verwendet um lesbare Daten des HTTP zu verschlüsseln und via TCP/IP zu versenden. Apache Tomcat bietet verschiedene „Connector“s um unterschiedliche Protokolle zu unterstützen. Für den Einsatz von SSL im Tomcat wird hier JSSE benutzt, da es standardmäßig in jedem JDK ab Version 5 enthalten ist und dadurch die Unterstützung für HTTPS bietet ohne zusätzliche Software nachinstallieren zu müssen. Hierzu wird ein Keystore angelegt und ein „Connector“ der Klasse „org.apache.coyote.http11.Http11Protocol“ für Port 8443 in die server.xml des Tomcats eingetragen. Die genaue Konfiguration lässt sich im Installationshandbuch oder in [Fou] nachlesen.

6.2.2. Server-Datenbank Kommunikation

Die Server-Datenbank Kommunikation muss nicht besonders geschützt werden, da Server und Datenbank auf dem selben Rechner arbeiten. Um die Kommunikation zu verwirklichen, wird eine Java Schnittstelle zu der PostgreSQL Datenbank benötigt, die möglichst gut auf die gewählte Architektur passt. PostgreSQL unterstützt alle vier Typen des JDBC (Java Database Connectivity, siehe auch Abschnitt 7.5.4) Treibers. Der JDBC Typ-4-Treiber bietet dem Programmierer eine einheitliche Schnittstelle zu allen unterstützen

Datenbanken, sodass ein späterer Datenbankwechsel durch den einfachen Austausch des Treibers und ohne Änderung des Programmcodes geschehen kann. Er kommuniziert direkt mit der Datenbank und übersetzt die JDBC Befehle in die datenbankspezifischen SQL-Anweisungen. Im System wird der JDBC Treiber von PostgreSQL in der aktuellsten Version 8.3-604 (siehe auch Abschnitt 7.5.4)) verwendet und durch die Konfiguration im Tomcat eingestellt, sodass Zugangsdaten jederzeit ohne Neukompilierung des Projektes geändert werden können.

7. Feinentwurf

Die Software wird nach dem Architekturmuster „Model-View-Controller“ programmiert. Dieses Muster trennt eine Anwendung in die drei Teilbereiche „Model“, „View“ und „Controller“ auf, wobei der Teilbereich „Model“ die Daten hält, der Bereich „View“ die Daten anzeigt und der Bereich „Controller“ den Programmfluss steuert. Im Kontext einer Webanwendung, sendet der Browser also eine HTTP-Anfrage an den Controller, der diesen auswertet, notwendige Aktualisierungen im Bereich „Model“ anstößt und schließlich die HTTP-Anfrage an den View weiterleitet, der die aktualisierten Daten des Modells ausliesst und eine HTTP-Antwort zurück an den Browser sendet. Abbildung 7.1 zeigt eine Grafik dieses Vorgangs. Der Vorteil dieses Vorgehens ist eine klare Trennung zwischen Daten, Kontrollfluss und Anzeige, sodass eines dieser Komponenten mit wenig Aufwand ausgetauscht werden kann. Dies ermöglicht z.B. verschiedene Benutzeroberflächen zu verwenden ohne etwas im Rest der Software verändern zu müssen.

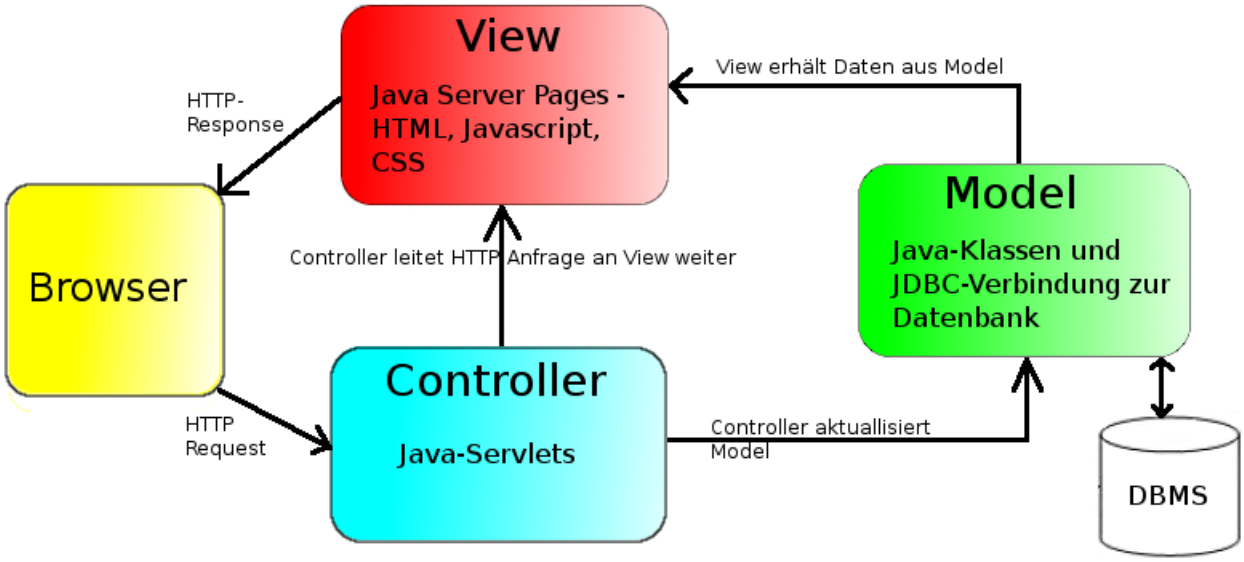

Abbildung 7.1.: Model-View-Controller

7.1. Paketstruktur

Die Paketstruktur orientiert sich am Model-View-Controller Architekturmuster und besteht aus den Paketen „system“, „model“ und „controller“. „system“ enthält Systemklassen für Datenbank und E-Mails (Abschnitt 7.4.2), „model“ enthält alle zum Datenmodell gehörenden Klassen (Abschnitt 7.2.2) und „controller“ enthält alle Java-Servlets (Abschnitt 7.4.3).

7.2. Datenmodell (model)

In den folgenden Abschnitten wird das Datenmodell entworfen. Das Vorgehen hierbei entspricht einem wichtigen Entwurfsverfahren der Informatik: Divide and Conquer. Dabei wird versucht ein Problem soweit in Teilprobleme zu zerlegen, bis jedes einzelne Teilproblem einfach lösbar ist. Des weiteren wird hier ein Objektorientierter Ansatz gewählt, indem jedes identifizierte Objekt einer Klasse entspricht. Erst hiernach wird überprüft ob das Problem minimierbar ist und einzelne Objekte in einer gemeinsamen Klasse zusammengefasst werden können.

7.2.1. Objekte und Beziehungen

Aus der Anforderungsanalyse kann man einige Objekte und deren Beziehung zueinander extrahieren. Abbildung 7.2 zeigt in einer Art Klassendiagramm eine Übersicht über die identifizierten Objekte. In Grün dargestellt sind die Objekte, die den Akteuren Student, Tutor, Dozent und Administrator entsprechen. Der Administrator besitzt für das Modell allerdings keine Bedeutung und wird daher im Modell nicht berücksichtigt. Im Abschnitt über Controller (Abschnitt 7.4) erhält er wieder eine Bedeutung und wird dort an anderer Stelle eingebaut. In Blau sind die Organisationseinheiten dargestellt. Zu ihnen gehören das Semester, die Veranstaltung, das Tutorium, die Übungsgruppe und die Termine. In Orange wurden schließlich die notengebenden Objekte eingefärbt. Zu ihnen gehören die Zwischennoten einer Veranstaltung, die Prüfungen, die Anmeldung zu Prüfungen, sowie die Endnote.

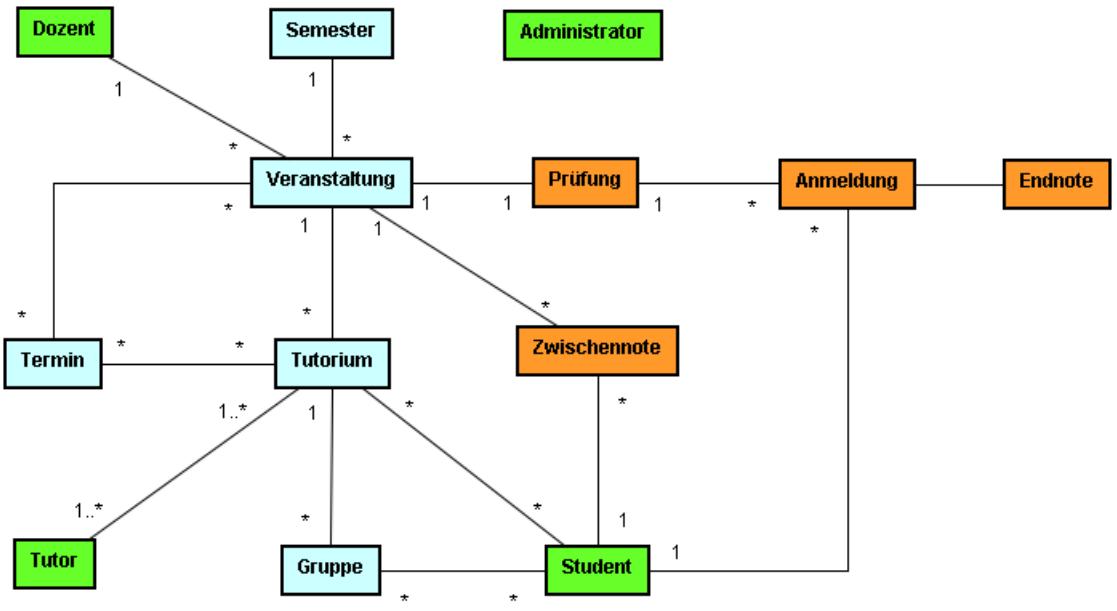

Abbildung 7.2.: Objekte und ihre Beziehungen

7.2.2. Subpakete von „model“

Im Folgenden wird aus den identifizierten Objekt ein detailliertes Klassenmodell erstellt, welches die funktionalen Anforderungen der jeweiligen Objekte sowie deren Beziehungen untereinander möglichst exakt umsetzt. Hierbei wird versucht ein einfaches Datenmodell zu erstellen, dass nur so komplex wie nötig ist. Redundanzen werden hier bewusst nicht vermieden, da die Klassen in der Software als Java Bean (Spezifikation: [SDNb]) verwendet werden und daher je nach Verwendungszweck die zusätzlichen Attribute benötigen. Würde dies nicht passieren, müsste zu viel Logik des Modells in den Controller oder die View verlagert werden. Zusätzlich werden alle Klassen mit „Setter“- und „Getter“-Methoden für alle ihre Attribute ausgerüstet, da die Klassen als Java-Bean dienen und alle Attribute als private deklariert werden da der Zugriff streng objektorientiert geschehen soll. Jegliche Arten von Arrays werden als Standardarray dargestellt. Dies soll allerdings nur darauf hinweisen, dass es sich hierbei um einen java.util.Collection-Typ handelt. In der Implementierung sollen also nicht zwingend Standardarrays benutzt werden.
Die Objekte lassen sich in drei verschiedene Pakete einteilen. Abbildung 7.3 zeigt die Paketstruktur des Modells. Die drei Pakete stellen Unterpakete des Pakets „model“ dar. Das Paket „user“ enthält alle Klassen, die den grünen Objekten des Objektdiagramms entsprechen. Die blauen Objekte kommen in das Paket „event“ und die orangefarbenen Objekte werden dem Paket „grade“ zugeordnet.

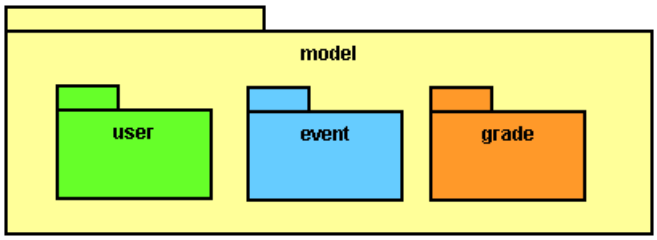

Abbildung 7.3.: Paketstruktur Model

7.2.3. Klassen im Paket „user“

Abbildung 7.4 zeigt einen Überblick über die erzeugten Klassen des Pakets „user“.

7.2.3.1. Klasse User

Die Klasse User entspricht dem Benutzer des Systems. Die Klasse dient der Benutzerverwaltung, sowie als Basisklasse für die Klassen Tutor, Academic und Student. In Abschnitt 7.4.1 wird näher auf die Authentifizierung und Zugriffskontrolle eingegangen.

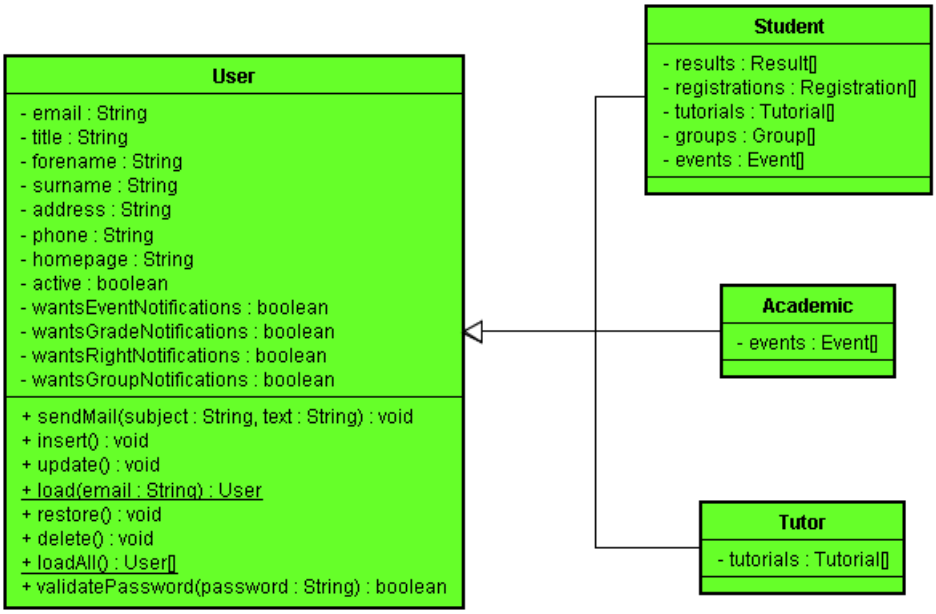

Abbildung 7.4.: Paket user

Attribute

Name	Typ	Beschreibung
email	String	E-Mailadresse des Benutzers
title	String	Titel des Benutzers z.B. „Dr.“ oder „Prof.“
forename	String	Vorname des Benutzers
surname	String	Nachname des Benutzers
address	String	Die komplette Anschrift des Benutzers
phone	String	Telefonnummer des Benutzers
homepage	String	Eine Homepage des Benutzers
active	boolean	Ist der Benutzeraccount aktiviert
wantsEventNotifications	boolean	Möchte Veranstaltungsbenachrichtigungen?
wantsGradeNotifications	boolean	Möchte Notenbenachrichtigungen?
wantsRightNotifications	boolean	Möchte Rechtebenachrichtigungen?
wantsGroupNotifications	boolean	Möchte Übungsgruppenbenachrichtigungen?

Funktionen

sendEmail(subject : String, text : String) : void

Sendet dem User eine E-Mail. Der Parameter subject entspricht dem Betreff der E-Mail. Der Parameter text entspricht dem Inhalt der E-Mail.

insert() : void

Fügt das aktuelle Userobjekt in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

load(email : String) : User (statisch)
Lädt einen Benutzer aus der Datenbank. Der Parameter email ist die E-Mailadresse des Benutzers, der geladen werden soll. Rückgabewert ist ein Objekt der Klasse User.

loadAll() : User[] (statisch)
Lädt alle Benutzer aus der Datenbank. Rückgabewert ist ein Array von Userobjekten.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht den aktuellen Benutzer in der Datenbank.

validatePassword(password : String) : boolean
Validiert das im Parameter password übergebene Passwort und gibt true zurück, wenn das Passwort richtig ist. Es werden hierfür die MD5-Hashwerte des eingegebenen und des in der Datenbank vorhandenen Passworts verglichen.

7.2.3.2. Klasse Academic

Die Klasse Academic entspricht dem Akteur Dozent. Sie wird von der Klasse User abgeleitet und erhält daher alle seine Attribute.

Attribute

Name	Typ	Beschreibung
events	Event[]	Eine Liste seiner Veranstaltungen

7.2.3.3. Klasse Tutor

Die Klasse Tutor entspricht dem Akteur Tutor. Sie wird von der Klasse User abgeleitet.

Attribute

Name	Typ	Beschreibung
tutorials	Tutorial[]	Eine Liste seiner Tutorien

7.2.3.4. Klasse Student

Die Klasse Student entspricht dem Akteur Student. Sie wird von der Klasse User abgeleitet.

Attribute

Name	Typ	Beschreibung
halfGrades	HalfGrade[]	Eine Liste seiner Zwischennoten
registrations	Registration[]	Eine Liste seiner Prüfungsanmeldungen
tutorials	Tutorial[]	Eine Liste seiner Tutorien
groups	Group[]	Eine Liste seiner Übungsgruppen
events	Event[]	Eine Liste seiner Veranstaltungen

7.2.4. Klassen im Paket „event“

Abbildung 7.5 zeigt einen Überblick über das Paket „event“.

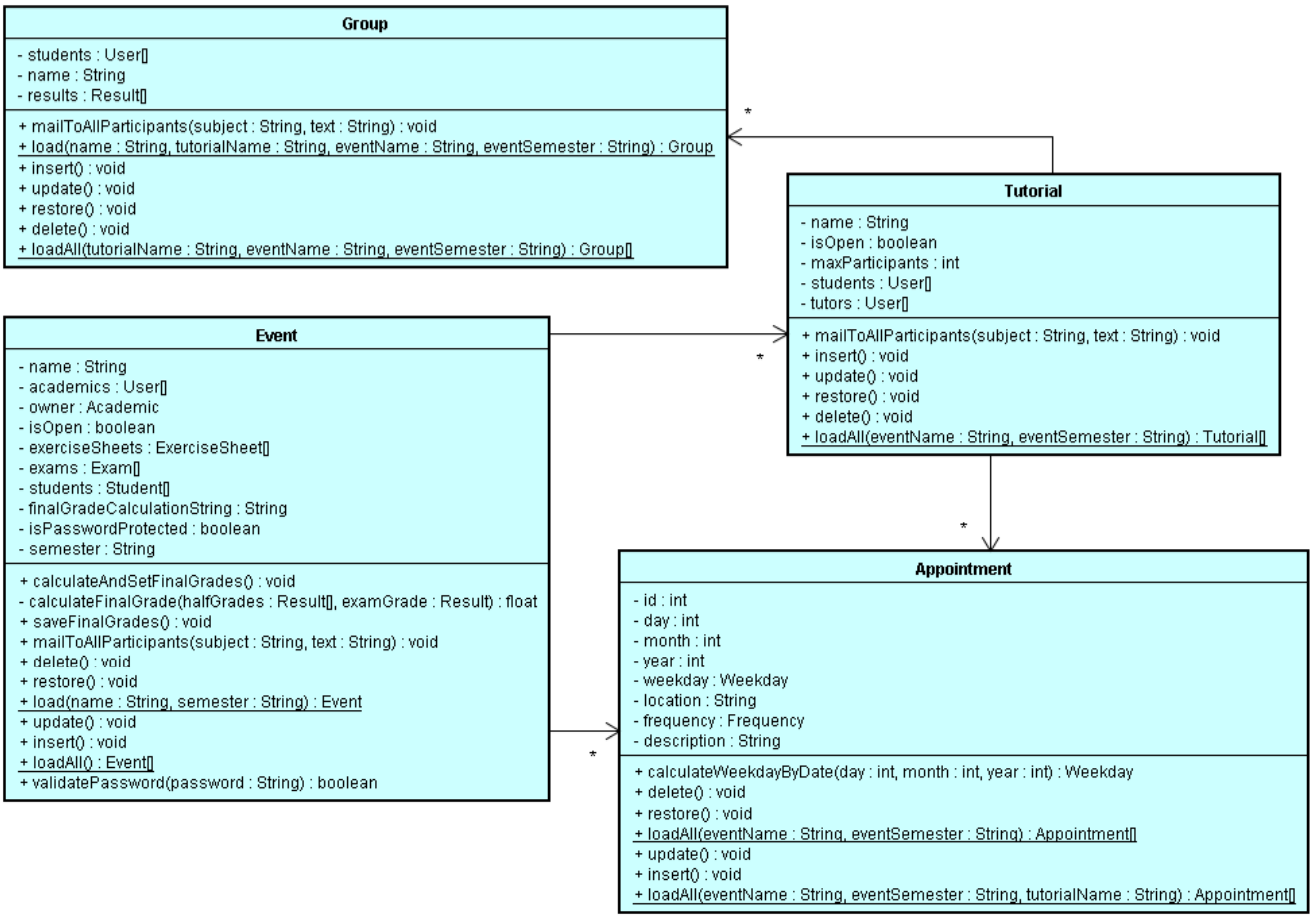

Abbildung 7.5.: Paket event

7.2.4.1. Klasse Semester

Ein Semester ist eine sehr einfache Klasse, die lediglich beliebig viele Veranstaltungen enthält. Ihr einziges Attribut ist eine eindeutige Bezeichnung, die beschreibt um welches zeitliche Semester es sich handelt. Da dies durch einen simplen String der Form „WS09/10“ oder „SS09“ für die Wintersemester 2009/2010 und Sommersemester 2009 verwirklicht werden kann, ist es sinnvoller eine HashMap zu verwenden, die den Bezeichner als Schlüssel verwendet und als Wert eine java.util.ArrayList von Veranstaltungen hält, als für das Semester eine eigene Klasse zu erstellen. Ein Vorteil ist, dass dadurch sichergestellt ist, dass ein Semester mit der Bezeichnung „SS09“ nur einmal vorhanden ist.

7.2.4.2. Klasse Event

Die Klasse Event entspricht der Veranstaltung.

Attribute

Name	Typ	Beschreibung
name	String	Name der Veranstaltung
semester	String	Bezeichnung des Semesters (Bsp.: „WS08/09“ oder „SS09“)
isOpen	boolean	Veranstaltung offen für Anmeldungen?
academics	User[]	Liste von Dozenten
owner	Academic	Der Ersteller der Veranstaltung
appointments	Appointment[]	Liste von Terminen
exams	Exam[]	Die Prüfungen
exerciseSheets	ExerciseSheet[]	Liste von Übungszetteln oder anderen Aufgaben
tutorials	Tutorial[]	Liste von Tutorien
students	Student[]	Liste von Studenten
finalGradeCalculationString	String	Die Zeichenfolge beschreibt wie die Endnotenberechnung durchgeführt wird. (siehe unten)
isPasswordProtected	boolean	Ist die Veranstaltung passwortgeschützt ?

Funktionen

calculateAndSetFinalGrades() : void
Errechnet die Endnoten der Studenten der Veranstaltung.

calculateFinalGrade(halfGrades : Result[], examGrade : Result) : float (private)
Errechnet die Endnote aus den übergebenen Zwischennoten im Parameter halfGrades und der Prüfungsnote im Parameter examGrade und gibt das Ergebnis als float-Wert zurück.

saveFinalGrades() : void
Speichert die Endnoten schließlich in der Datenbank.

mailToAllParticipants(subject : String, text : String) : void

Sendet allen Teilnehmern der Veranstaltung eine E-Mail. Der Parameter subject entspricht dem Betreff der E-Mail. Der Parameter text entspricht dem Inhalt der E-Mail.

insert() : void
Fügt das aktuelle Eventobjekt in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

load(name : String, semester : String) : Event (statisch)
Lädt eine Veranstaltung aus der Datenbank. Der Parameter name entspricht dem Namen der Veranstaltung. Der Parameter semester entspricht dem Semester der Veranstaltung. Rückgabewert ist ein Objekt der Klasse Event.

loadAll() : Event[] (statisch)
Lädt alle Veranstaltungen aus der Datenbank. Rückgabewert ist ein Array von Eventobjekten.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht die aktuelle Veranstaltung in der Datenbank.

validatePassword(password : String) : boolean
Validiert das im Parameter password übergebene Passwort und gibt true zurück, wenn das Passwort richtig ist.

Syntax des Berechnungsstrings „finalGradeCalculationString“

Die Syntax der Zeichenfolge, die in finalGradeCalculationString gespeichert werden soll, wird durch die folgende Grammatik (in Erweiterter Backus-Naur-Form, siehe auch [Wik09]) beschrieben:

finalCalculationString = Operand [Rechenzeichen Operand]* ;
Operand = Zahl | Parameter | ”(” finalCalculationString ”)” ;
Parameter = ”p” Zahl ;
Zahl = [Ziffer]+ ;
Ziffer = ”0” | ”1” | ”2” | ”3” | ”4” | ”5” | ”6” | ”7” | ”8” | ”9” ;
Rechenzeichen = ”+” | ”-” | ”*” | ”/” ;

Die Speicherung des Berechnungsalgorithmus für die Endnoten in einer Zeichenfolge bringt den Vorteil, dass auch ein Mensch den Algorithmus durch lesen verstehen kann. Es ist möglich auch ohne eine entspre-

chende Benutzungsoberfläche eine Berechnung einzustellen. Die Berechnung findet durch die Benutzung von Operanden statt, welche durch Rechenzeichen miteinander verknüpft werden. Ein Operand ist eine Zahl oder ein Parameter, wobei mit Parameter eine Zwischennote der Veranstaltung gemeint ist. Am besten lässt sich dieses Verfahren durch ein Beispiel verdeutlichen: Man nehme an, eine Veranstaltung besitzt zwei Zwischennoten (z.B. Übungszettel) und eine Klausurnote. Jede dieser Zwischennoten erhält eine Parameternummer, welche durch die Benutzungsoberfläche angezeigt wird. Nun möchten wir die Endnote aus dem Durchschnitt der Zwischennoten und der Klausurnote errechnen. Hierraus ergibt sich folgende Zeichenfolge: „(p0 + p1 + p2) / 3“.

7.2.4.3. Klasse Tutorial

Die Klasse Tutorial entspricht einem Tutorium.

Attribute

Name	Typ	Beschreibung
name	String	Name des Tutoriums
maxParticipants	int	Maximale Anzahl an Teilnehmern
isOpen	boolean	Tutorium offen für Anmeldungen?
groups	Group[]	Liste von Übungsgruppen
students	User[]	Liste von Studenten
tutors	User[]	Liste von Tutoren

Funktionen

mailToAllParticipants(subject : String, text : String) : void
Sendet allen Teilnehmern des Tutoriums eine E-Mail. Der Parameter subject entspricht dem Betreff der E-Mail. Der Parameter text entspricht dem Inhalt der E-Mail.

insert() : void
Fügt das aktuelle Tutorialobjekt in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht das aktuelle Tutorium in der Datenbank.

loadAll(eventName : String, eventSemester : String) : Tutorial[] (statisch)
Lädt alle Tutorien einer Veranstaltung aus der Datenbank. eventName entspricht dem Namen, eventSemester dem Semester der Veranstaltung.

7.2.4.4. Klasse Appointment

Appointment entspricht einem Termin.

Attribute

Name	Typ	Beschreibung
day	int	Der Tag des Monats
month	int	Der Monat als Zahl beginnend mit 0 für Januar
year	int	Das Jahr (vierstellig)
weekday	Weekday	Der Wochentag
location	String	Ort/Raum
frequency	Frequency	Regelmäßigkeit
description	String	Zusätzliche Beschreibung

Enums

Name	Werte	Beschreibung
Weekday	monday,tuesday,wednesday,thursday,friday,saturday,sunday	Wochentage
Frequency	weekly,biweekly,monthly,once	Regelmäßigkeiten

Funktionen

calculateWeekdayByDate(day : int, month : int, year : int) : Weekday
Errechnet den Wochentag eines Datums. Die Parameter day, month und year entsprechen dem Datum, dessen Wochentag berechnet werden soll.

insert() : void
Fügt das aktuelle Terminobjekt in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht den aktuellen Termin in der Datenbank.

loadAll(eventName : String, eventSemester : String) : Appointment[] (statisch)
Lädt alle Termine einer Veranstaltung aus der Datenbank. eventName entspricht dem Namen, eventSemester dem Semester der Veranstaltung. Rückgabewert ist ein Array von Objekten der Klasse Appointment.

loadAll(eventName : String, eventSemester : String, tutorialName : String) : Appointment[] (statisch)
Lädt alle Termine eines Tutoriums aus der Datenbank. eventName entspricht dem Namen, eventSemester dem Semester der Veranstaltung. tutorialName entspricht dem Namen des Tutoriums. Rückgabewert ist ein Array von Objekten der Klasse Appointment.

7.2.4.5. Klasse Group

Die Klasse Group entspricht der Übungsgruppe.

Attribute

Name	Typ	Beschreibung
name	String	Name der Übungsgruppe (Bsp: „Übungsgruppe 1“)
students	Student[]	Liste der Studenten
halfGrades	HalfGrade[]	Liste der Zwischennoten

Funktionen

mailToAllParticipants(subject : String, text : String) : void
Sendet allen Teilnehmern der Übungsgruppe eine E-Mail. Der Parameter subject entspricht dem Betreff der E-Mail. Der Parameter text entspricht dem Inhalt der E-Mail.

insert() : void
Fügt das aktuelle Übungsgruppenobjekt in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht die aktuelle Übungsgruppe in der Datenbank.

loadAll(tutorialName : String, eventName : String, eventSemester : String) : Group[] (statisch)
Lädt alle Übungsgruppen eines Tutoriums aus der Datenbank. eventName entspricht dem Namen, eventSemester dem Semester der Veranstaltung. tutorialName entspricht dem Namen des Tutoriums. Rückgabewert ist ein Array von Objekten der Klasse Group.

7.2.5. Klassen im Paket „grade“

Abbildung 7.6 zeigt einen Überblick über das Paket „grade“.

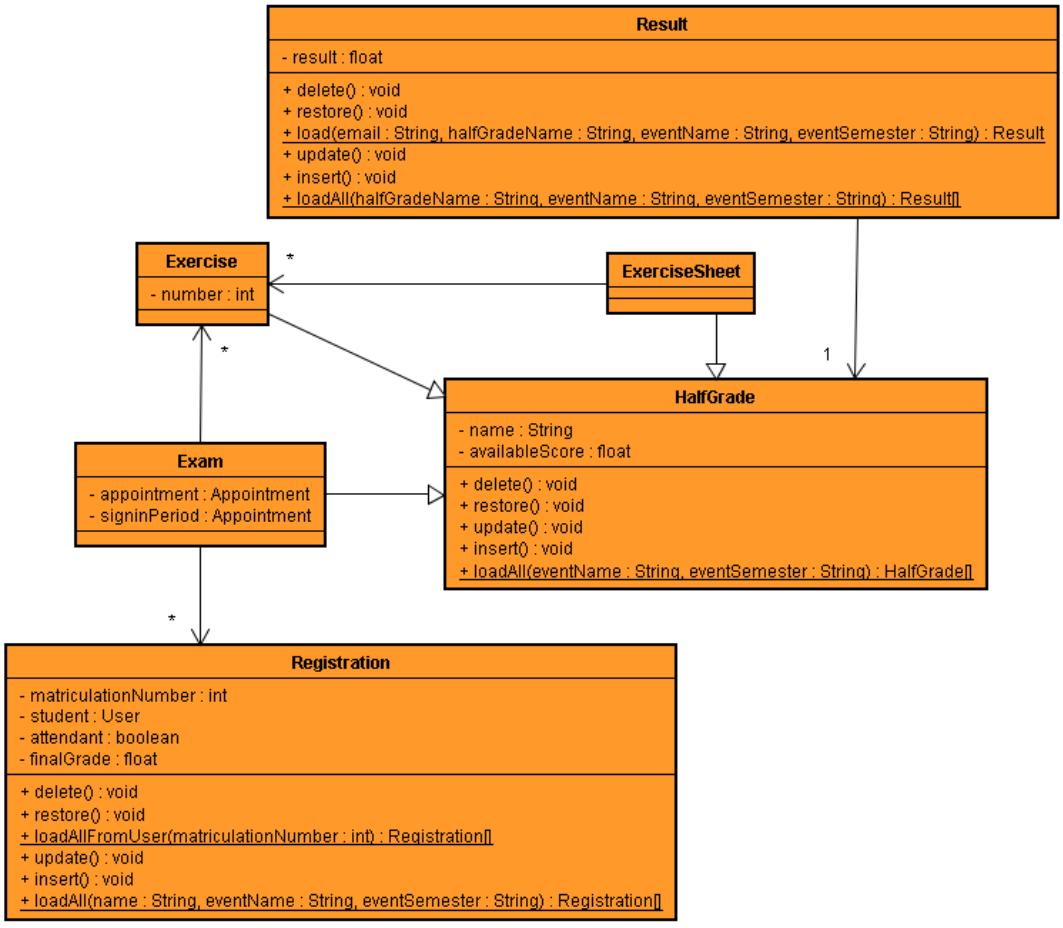

Abbildung 7.6.: Paket grade

7.2.5.1. Klasse HalfGrade

Die Klasse HalfGrade entspricht den Zwischennoten. Sie dient als Basisklasse von den Klassen Exam, ExerciseSheet und Exercise.

Attribute

Name	Typ	Beschreibung
name	String	Eine Beschreibung der Zwischennote
availableScore	float	Die maximale Punktzahl, die erreicht werden kann

Funktionen

insert() : void

Fügt die aktuelle Zwischennote in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht die aktuelle Zwischennote in der Datenbank.

loadAll(eventName : String, eventSemester : String) : HalfGrade[] (statisch)
Lädt alle Zwischennoten eines Tutoriums aus der Datenbank. eventName entspricht dem Namen, eventSemester dem Semester der Veranstaltung. Rückgabewert ist ein Array von Objekten der Klasse HalfGrade.

7.2.5.2. Klasse Exam

Die Klasse Exam entspricht den Prüfungen einer Veranstaltung. Sie erbt die Attribute und Funktionen von der Klasse HalfGrade.

Attribute

Name	Typ	Beschreibung
appointment	Appointment	Der Termin der Prüfung
signinPeriod	Appointment	Der Anmeldezeitraum
exercises	Exercise[]	Eine Liste von Aufgaben
registrations	Registration[]	Eine Liste der Anmeldungen zur Prüfung

7.2.5.3. Klasse ExerciseSheet

Die Klasse ExerciseSheet entspricht den Übungszetteln der Veranstaltung. Die Klasse kann aber durchaus auch verwendet werden um andere Arten von Zwischennoten zu repräsentieren. Sie erbt die Attribute und Funktionen von der Klasse HalfGrade.

Attribute

Name	Typ	Beschreibung
exercises	Exercise[]	Eine Liste von Aufgaben

7.2.5.4. Klasse Exercise

Die Klasse Exercise entspricht den Aufgaben einer Prüfung oder eines Übungszettels. Sie erbt die Attribute und Funktionen von der Klasse HalfGrade.

Attribute

Name	Typ	Beschreibung
number	int	Für die Nummerierung von Aufgaben

7.2.5.5. Klasse Registration

Die Klasse Registration entspricht einer Anmeldung zu einer Prüfung.

Attribute

Name	Typ	Beschreibung
matriculationNumber	int	Die Matrikelnummer des Studenten, der sich anmeldet
student	Student	Der Student
attendant	boolean	War der Student bei der Prüfung anwesend?
finalGrade	float	Die Endnote des Studenten

Funktionen

insert() : void
Fügt die aktuelle Anmeldung in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht die Anmeldung in der Datenbank.

loadAll(name : String, eventName : String, eventSemester : String) : Registration[] (statisch)
Lädt alle Anmeldungen zu einer Prüfung aus der Datenbank. name entspricht dem Namen der Prüfung, eventName entspricht dem Namen und eventSemester dem Semester der Veranstaltung. Rückgabewert ist ein Array von Objekten der Klasse Registration.

loadAllFromUser(matriculationNumber : int) : Registration[] (statisch)
Lädt alle Anmeldungen eines Studenten aus der Datenbank. matriculationNumber entspricht der Matrikelnummer des Studenten. Rückgabewert ist ein Array von Objekten der Klasse Registration.

7.2.5.6. Klasse Result

Die Klasse Result entspricht einem Ergebnis einer Prüfung, eines Übungszettels oder einer Aufgabe eines Studenten.

Attribute

Name	Typ	Beschreibung
result	float	Ergebnis
correspondingHalfGrade	HalfGrade	Referenz auf die Zwischennote für die das Ergebnis gilt.

Funktionen

insert() : void
Fügt das aktuelle Ergebnis in die Datenbank ein.

update() : void
Die Funktion speichert den aktuellen Status des Objektes in die Datenbank.

restore() : void
Stellt den Datenbankzustand im Objekt wieder her.

delete() : void
Löscht das aktuelle Ergebnis in der Datenbank.

loadAll(halfGradeName : String, eventName : String, eventSemester : String) : Result[] (statisch)
Lädt alle Ergebnisse einer Zwischennote einer Veranstaltung aus der Datenbank. halfGradeName entspricht dem Namen der Zwischennote, eventName entspricht dem Namen und eventSemester dem Semester der Veranstaltung. Rückgabewert ist ein Array von Objekten der Klasse Result.

load(email : String, halfGradeName : String, eventName : String, eventSemester : String) : Result (statisch)
Lädt ein Ergebnis eines Studenten einer Zwischennote aus der Datenbank. email entspricht der E-Mail des Studenten, halfGradeName entspricht dem Namen der Zwischennote, eventName entspricht dem Namen und eventSemester dem Semester der Veranstaltung.

7.3. Grafische Benutzeroberfläche (View)

Nachdem im vorherigen Kapitel das „Backend“ der Software beschrieben wurde, geht dieser Abschnitt auf das „Frontend“ ein: die grafische Benutzeroberfläche. Beschrieben wird im Folgenden wie die Oberfläche der Seite aufgebaut sein soll, die später im Browser zu sehen ist. Des Weiteren wird auf die dynamische Komponente der Seite eingegangen, die durch Javascript programmiert werden soll. Abbildung 7.7 zeigt die Einteilung der Seite. Unter -1- sieht man stets das Menü mit den aktuellen Menüoptionen des jeweiligen Benutzers. Unter -2- sieht man den Bereich des Hauptinhalts. Hier werden alle Informationen angezeigt. In diesem Bereich sollen viele Inhalte durch AJAX nachgeladen werden. Dies soll eine schnelle und angenehme Interaktion mit der Webseite ermöglichen. Stellt man sich beispielsweise vor, dass jedesmal wenn eine Note eingetragen wird ohne AJAX ein HTTP-Request das Nachladen der kompletten Webseite erzwingt, würde dies die Interaktionsgeschwindigkeit mit der Webseite stark einschränken. Listen sollen grundsätzlich beim Anklicken von Einträgen sofort weitere Informationen liefern. Geordnet sind diese

Listen standardmäßig immer aufsteigend. Sollte es sich anbieten, kann während der Implementierung aber eine Funktion zur Verfügung gestellt werden, die es dem Benutzer erlaubt die Sortierung zu verändern. Dort wo es sinnvoll ist, sollen Werte direkt in den Listen verändert werden können (z.B. Noteneintragung). Jegliche Änderungen werden, erst übernommen wenn die entsprechende Option (Submit) ausgewählt wurde. Unter -3- sieht man einen Rahmen der Universität, der immer sichtbar sein soll. -4- ist ein Bereich indem Systemhinweise (z.B. „Login fehlgeschlagen“) sichtbar werden sollen.

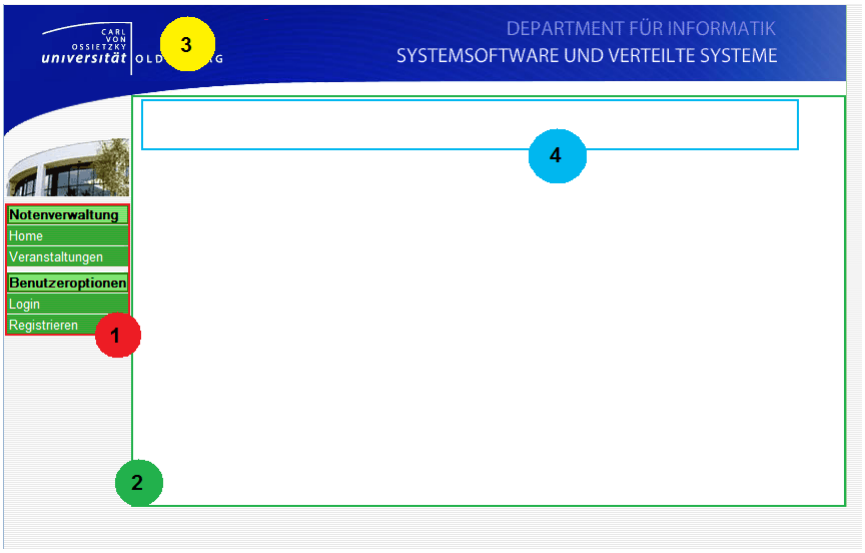

Abbildung 7.7.: Einteilung der Seitenoberfläche

7.3.1. Java Server Pages und Pageflow

Im folgenden werden die nötigen JSP’s erfasst, die zum Anzeigen und Bearbeiten der Informationen im System verwendet werden. Hierbei wird auch auf die Zugriffsrechte der einzelnen Akteure eingegangen, wobei die in Abschnitt 3 angesprochene Hierarchie zum Einsatz kommt. Alles was ein Tutor der Veranstaltung darf, darf auch der Dozent und der Administrator. Die Aufgabenbereiche werden sehr modular in viele JSP’s aufgeteilt, damit keine Redundanzen in dem Anzeigen gleichartiger Informationen auf verschiedenen Seiten entstehen. Folgende JSP’s werden benötigt:

home.jsp: Stellt einen begrüßenden Text dar, der das System beschreibt oder aktuelle Nachrichten enthält.

register.jsp: Stellt ein Formular zum Registrieren zur Verfügung indem alle Daten der Klasse User eintragbar sind. Im Falle einer erfolgreichen Registrierung wird ein Text angezeigt der den Benutzer darauf hinweisst, dass er seinen Account nun aktivieren muss. Im Fehlerfall wird das Registrierungsformular wieder angezeigt und ein Text eingeblendet, der den Fehler beschreibt.

passwordrecovery.jsp: Auf dieser Seite kann der Benutzer seine E-Mailadresse eingeben und ein neues Passwort anfordern. Ob der Vorgang erfolgreich war oder nicht, erfährt der Benutzer durch einen eingeblendeten Text.

login.jsp: Stellt ein Formular zur Verfügung indem man E-Mail und Passwort eingeben muss um sich anzumelden. Außerdem gibt es einen Link zu passwordrecovery.jsp und register.jsp. Ist der Anmeldevorgang erfolgreich, so wird auf die Seite userhome.jsp weitergeleitet. Im Falle eines Fehlers wird das Login Formular wieder angezeigt und ein Hinweis eingeblendet.

usersettings.jsp: Ist vom Aufbau ähnlich wie register.jsp, allerdings gibt es eine Möglichkeit das Passwort zu ändern nachdem man das alte Passwort eingegeben hat. Im Erfolgsfall wird der Benutzer auf userhome.jsp umgeleitet und durch einen eingeblendeten Text über den Erfolg des Vorgangs informiert. Im Fehlerfall bleibt der Benutzer auf der Seite und bekommt einen Hinweis eingeblendet, was fehlgeschlagen ist. Die E-Mailadresse, der Vorname und der Nachname lassen sich nicht ändern, da hierauf die Zugriffskontrolle und Notengebung beruht (siehe Abschnitt 7.4.1).

useradministration.jsp: Ist nur durch Administratoren betretbar. Hier werden alle registrierten Benutzer des Systems angezeigt. Der Administrator hat hier die Möglichkeit einen Benutzer zu erstellen, zu löschen oder zu ändern. Er kann hier auch Dozenten und Administratoren ernennen.

studentgradelist.jsp: Diese Seite stellt eine Liste von Studenten dar. Es stellt alle Ergebnisse des jeweiligen Studenten in der gerade angezeigten Veranstaltung dar und bietet Optionen diese direkt zu ändern. Die Seite kann nur durch Tutoren oder Dozenten der Veranstaltung und durch Administratoren eingesehen werden.

academiclist.jsp: Diese Seite stellt eine Liste aller Dozenten der jeweiligen Veranstaltung dar. Hier soll auch ersichtlich sein, welcher Dozent die Veranstaltung besitzt.

appointment.jsp: Hier wird ein Termin angezeigt. Er kann durch den Dozenten der jeweiligen Veranstaltung verändert werden.

appointmentlist.jsp: Diese Seite zeigt eine Liste aller Termine einer Veranstaltung oder eines Tutoriums an. Dozenten können hier einzelne Termine löschen oder hinzufügen. Per Klick auf einen Termin wird mithilfe von appointment.jsp der entsprechende Termin angezeigt.

groupchoice.jsp: Auf dieser Seite, kann ein Student seine Wunschgruppe angeben. Dazu gibt er die E-Mailadressen seiner Wunschpartner ein.

tutorialchoice.jsp: Hier kann ein Student seine Wunschtutorien angeben. Hierzu kann er seinen Erstwunsch, Zweitwunsch und Drittwunsch angeben.

group.jsp: Auf dieser Seite sollen Informationen zu einer ausgewählten Übungsgruppe angezeigt werden. Daten die hier angezeigt werden, sollen durch Dozenten und Tutoren der Veranstaltung änderbar sein.

grouplist.jsp: Hier wird eine Liste aller Übungsgruppen der Veranstaltung angezeigt. Bei einem Klick auf eine Übungsgruppe sollen mithilfe von group.jsp die verfügbare Informationen zu der Übungsgruppe sichtbar werden. Einzelne Gruppen lassen sich hier löschen, hinzufügen oder ändern. Außerdem kann einer Übungsgruppe ein Student hinzugefügt oder entfernt werden.

tutorial.jsp: Diese JSP zeigt alle Tutoriumsinformationen an und bietet Möglichkeiten um diese direkt durch Dozenten oder den jeweiligen Tutor zu ändern.

tutoriallist.jsp: Hier wird eine Liste aller Tutorien einer bestimmten Veranstaltung angezeigt. Als Optionen, soll man hier per Klick auf ein Tutorium eine Informationsseite durch tutorial.jsp angezeigt bekommen. Außerdem soll man als Dozent der Veranstaltung Tutorien hinzufügen, löschen und ändern, sowie Studenten zum Tutorium hinzufügen oder entfernen können.

event.jsp: Zeigt alle Informationen über eine Veranstaltung an. Die Seite verwendet tutoriallist.jsp um Tutorien, studentlist.jsp um angemeldete Studenten und academiclist.jsp um Dozenten anzuzeigen. Änderungsoptionen werden nur angezeigt, wenn der aktuelle Benutzer über entsprechende Rechte verfügt. Dozenten sowie Administratoren haben hier die Möglichkeit alle Veranstaltungsdaten zu ändern, Studenten zu entfernen oder Dozenten für die Veranstaltung zu ernennen. Diese sind dann eingeschränkte Dozenten für diese Veranstaltung. Außer der Option neue Dozenten zu ernennen oder Dozenten zu entfernen haben sie alle Optionen, die auch der Dozent hat.

eventlist.jsp: Stellt eine Liste vorhandener Veranstaltungen bereit. Diese Liste muss nicht alle Veranstaltungen enthalten. Bei einem Klick auf eine Veranstaltung soll event.jsp sichtbar werden. Administratoren dürfen hier Veranstaltungen entfernen, wodurch allerdings auch alle Noten verloren gehen. Dozenten dürfen neue Veranstaltungen hinzufügen.

exercisesheet.jsp: Hier gibt es die Möglichkeit neue Übungszettel einzutragen, oder vorhandene Daten zu bearbeiten. Schreibenden Zugriff hat nur der Dozent der Veranstaltung. Alle anderen Akteure können die Informationen abfragen. Hier wird auch eine Liste von Übungsaufgaben angezeigt und verwaltet.

exercisesheetlist.jsp: Diese Seite zeigt alle Übungszettel der Veranstaltung an. Bei einem Klick auf einen Übungszettel soll mithilfe von exercisesheet.jsp Informationen des Übungszettels angezeigt und bearbeitet werden können. Des weiteren soll es eine Möglichkeit geben Übungszettel aus einer anderen Veranstaltung zu übernehmen.

exam.jsp: Auf dieser Seite ist es möglich Informationen über eine Prüfung einzusehen. Dozenten können hier zusätzlich sehen, welche Studenten sich zur Prüfung angemeldet haben und können bei diesen Studenten Noten für die Prüfung eintragen.

examslist.jsp: Hier wird eine Liste von Prüfungen angezeigt. Dozenten der Veranstaltung können hier neue Prüfungen anlegen, alte Prüfungen entfernen oder durch einen Klick auf eine Prüfung über exam.jsp Änderungen an einer Prüfung vornehmen. Außerdem soll es möglich sein Prüfungen aus anderen Veranstaltungen zu übernehmen.

registertoexam.jsp: Ein Student kann sich hier durch Eingabe seiner Matrikelnummer und seines Benutzerpassworts zu einer Prüfung anmelden sofern die Anmeldung innerhalb des Anmeldezeitraums geschieht.

finalgradecalculation.jsp: Hier wird eingestellt, wie die Endnote der Veranstaltung berechnet werden soll. Es soll eine leicht verständliche Oberfläche entstehen. Zusätzlich soll man die Berechnung aber auch per Eingabe einer Berechnungszeichenfolge (siehe Abschnitt 7.2.4.2) einstellen können.

studentgradelist.jsp: In dieser Liste werden Studenten einer Veranstaltung mit ihren Noten aufgelistet. Änderungen sollen direkt eingetragen werden können, wobei der Dozent alle Zwischennoten eintragen kann und der Tutor nur die Zwischennoten der Studenten seines Tutoriums. Eine durch das System errechnete Endnote kann hier überschrieben werden.

userhome.jsp: Dies ist die Startseite eines angemeldeten Benutzers. Er erhält hier eine Übersicht über seine Veranstaltungen, Tutorien, Übungsgruppen und Noten.

index.jsp Stellt den Rahmen der Seite zur Verfügung. Hier wird das Menü und die Informationsseite eingefügt um den Oberflächenentwurf (Abbildung 7.7) umzusetzen. Außerdem wird hier auch der Bereich für die Hinweise eingefügt.

7.3.2. Javascript und AJAX

Javascript und im Speziellen das AJAX Konzept ist heutzutage von einer professionellen Webseite kaum noch wegzudenken. Es bietet Vorteile sowohl in der Benutzerführung als auch in der Performance auf den dargestellten Webseiten. Diese Vorteile sollen auch in diesem Projekt durch die Wahl eines AJAX-Frameworks ausgenutzt werden, sodass im Folgenden stichwortartig ein kleiner Überblick über aktuelle AJAX-Frameworks gegeben wird. Es soll im Bezug auf das aktuelle Projekt eine kleine Liste von Pro- und Contra-Argumenten entstehen, sodass anschließend eine Auswahl eines Frameworks getroffen werden kann.

7.3.3. AJAX Frameworks

Alle folgenden Frameworks bieten gute AJAX-Funktionalitäten. Für das Projekt wären jedoch außer AJAX noch andere Funktionalitäten interessant. AJAX-fähige Tabs wären gut um zwischen verschiedenen Inhalten einer Veranstaltung hin und her zuspringen (z.B. Tutorials anzeigen, Studenten anzeigen). Accordions wären gut um Semester ein und auszublenden. Außerdem wäre eine Möglichkeit zur direk-

ten Bearbeitung von Listeneinträgen sinnvoll (z.B. Noteneintragung). Sortierbare Listen[1] wären ebenfalls wünschenswert. Eine komfortable Möglichkeit ein Datum und eine Uhrzeit auszuwählen, wäre bei der Terminerstellung interessant. Betrachtet werden die Frameworks „Dojo Toolkit“ (Version 1.2.3, `http://dojotoolkit.org/`), „Scriptaculous“ (Version 1.8.2, `http://script.aculo.us/`), „jQuery“ (Version 1.3.2, `http://www.jquery.com/`) und „Mootools“ (Version 1.2.1, `http://www.mootools.net/`).

Framework	Dojo Toolkit	Scriptaculous	jQuery	Mootools
Geschwindigkeit	sehr langsam	durchschnitt	durchschnitt	sehr schnell
Tabs	ja	ja	ja	nein
Accordions	ja	es gibt Communityimplementierungen	ja	ja
sortierbare Listen	ja	nein	ja	nein
Kalendar	ja	nein	ja	nein
Kommentar	sehr umfangreich	-	viele PlugIns	-

Da jQuery alle gewünschten Funktionalitäten unterstützt und um einiges schneller als das Dojo Toolkit ist, wird es in diesem Projekt verwendet.[2]

7.4. Funktionales Modell (Controller)

Aus dem vorherigen Abschnitt kann man nun die benötigten Controller und Funktionen identifizieren. Zunächst wird festgelegt, wie die Zugriffskontrolle und Authentifizierung der Benutzer durchgeführt wird. Danach werden zwei Manager Klassen entworfen, die für die Verwaltung der Datenbank und des E-Mailsystems zuständig sind. Die eigentlichen Controller sind die Servlets, die hiernach entworfen werden.

7.4.1. Zugriffskontrolle und Authentifizierung

Beim Anmelden in das System wird eine Session erstellt und in diesem die Zugriffsrechte des Benutzers gespeichert. Somit erhält die Session folgende Attribute:

Attribute der Session

Name	**Typ**	**Beschreibung**
isAdmin	boolean	Ist der Benutzer ein Administrator?
isAcademic	boolean	Ist der Benutzer ein Dozent?
isTutor	boolean	Ist der Benutzer ein Tutor?

Durch diese Attribute kann bestimmt werden, über welche Rechte der Benutzer im System auf der aktuell angezeigten Seite verfügt. isAcademic und isTutor hat hierbei allerdings nur auf userhome.jsp einen Nutzen, da hieraus abgeleitet wird ob der Benutzer eigene Tutorien oder Veranstaltungen besitzt, die angezeigt werden können. Ob der Benutzer entsprechende Änderungsrechte in einem Tutorium oder einer

[1] Hiermit ist eine Liste gemeint, die z.B. durch Anklicken einer Spalte nach dem Alphabet oder einer anderen Reihenfolge sortiert werden kann.

[2] Angaben basieren auf eigenen Versuchen. Besonders die tatsächliche Geschwindigkeit der Frameworks kann von den hier angegebenen Werten abweichen.

Veranstaltung besitzt wird durch einen Vergleich der E-Mailadresse des aktuellen Benutzers und der Tutoren bzw. Dozenten der Veranstaltung bestimmt. Da E-Mailadressen nachträglich nicht geändert werden können und einzigartig sind, ist dies eine sichere Authentifizierungsmethode. Ist isAdmin true, so hat der Benutzer auf jeder Seite alle möglichen Rechte.

7.4.2. Manager Klassen

Im Folgenden werden zwei zentrale Klassen zur Steuerung der Datenbank und des E-Mailsystems vorgestellt. Sie werden dem Paket „system“ zugeordnet. Beide Klassen entsprechen dem Entwurfsmuster „Fassade“. Dies vereinfacht den Zugriff auf die beiden Subsysteme „Datenbank“ und „E-Mail“ durch das Datenmodell oder die Controller.

7.4.2.1. Datenbankmanager

Zur Verwaltung der Datenbank macht es Sinn eine eigene Klasse zu verwenden, die alle Datenbankoperationen vornimmt. Angestoßen werden diese durch den Aufruf von den Funktionen delete, restore, load, insert und update der Klassen des Modells. Die Datenbankmanager Klasse selbst verwendet das Entwurfsmuster Singleton sodass jede Datenbankoperation von einem einzelnen Objekt verwaltet wird, auf welches man von überall im Programmcode Zugriff hat. Die Klasse sorgt außerdem dafür, dass Änderungen an Veranstaltungen und Noten mitgeloggt werden. Außerdem verwendet die Klasse die Klasse PGPoolingDataSource um Connectionpooling zur Datenbank zu unterstützen, da der Verbindungsaufbau die teuerste Operation ist. SQL-Statements werden per PreparedStatement ausgeführt, da diese vorkompiliert und im Cache gespeichert werden. Ein späteres Zugreifen auf das gleiche Statement wird hierdurch schneller. Zusätzlich sorgt diese Klasse dafür, dass alle Änderungen am Datenbankzustand mitgeloggt werden, sodass später genau verfolgt werden kann, wer welche Änderungen vorgenommen hat. In Abschnitt 7.5.2 wird hierfür eine Tabelle „Systemlog“ eingeführt.

Zugriff auf die Datenbank Der Zugriff auf die Daten erfolgt immer über die Klassen des Modells. Sie dienen als Schnittstelle zur Klasse DatabaseManager. Der Controller stellt hierbei eine Anfrage an die entsprechende Klasse indem es eine der Datenbankfunktionen aufruft. Die Klasse ruft nun die entsprechende Funktion im DatabaseManager auf. Der Vorteil dieser Vorgehensweise ist, dass der DatabaseManager abgekapselt vom Controller ist und somit Änderungen an der Klasse DatabaseManager nur in der entsprechenden Klasse des Modells übernommen werden müssen. Abbildung 7.8 zeigt dieses Vorgehen anhand eines Beispiels mithilfe eines Sequenzdiagramms.

7.4.2.2. EmailManager

Die Benachrichtigung durch E-Mails ist ein zentraler Service im System. Diverse Situationen führen dazu, dass E-Mails versendet werden. Es gibt hierfür vier Typen von Benachrichtigungen: Benachrichtigung bei...

- ...Veranstaltungsänderung:
  Ist man in einer Veranstaltung angemeldet und Änderungen werden hier vorgenommen (z.B. der

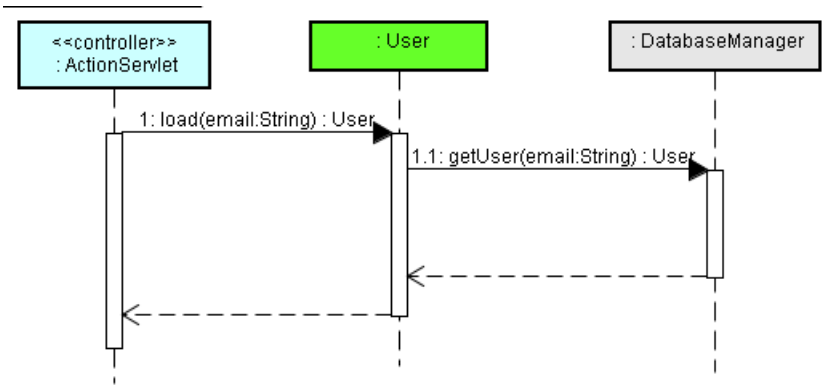

Abbildung 7.8.: Sequenzdiagramm Datenbankzugriff (Beispiel User.load())

Name der Veranstaltung wird geändert oder ein Tutorium wird hinzugefügt), so erhält jeder eingetragene Benutzer eine E-Mail, solange er dies nicht verbietet.

- ...Notenänderung:
  Wird eine eigene Note oder Zwischennote verändert oder eingetragen, so wird man benachrichtigt, solange man dies nicht verboten hat.

- ...Änderung der Rechte:
  Wird der eigene Status verändert (z.B. Ernennung zum Dozenten oder Tutor oder das Entfernen des Dozenten Status), so erhält man eine Benachrichtigung, solange man dies nicht verboten hat.

- ...Gruppenänderung:
  Finden Änderungen in eigenen Übungsgruppen statt (z.B. verlässt ein Student die Gruppe oder man wird einer neuen Gruppe zugeteilt), so erhält man eine Nachricht, solange man dies nicht verboten hat.

Die Klasse EmailManager soll das E-Mailsystem verwalten. Über diese Klasse ist es möglich Mails zu versenden. Wie beim DatabaseManager verwendet die Klasse das Entwurfsmuster Singleton, sodass es von überall zugreifbar ist und die alleinige Kontrolle über das E-Mailsystem hat. Abbildung 7.9 zeigt das Klassendiagramm der Klasse.

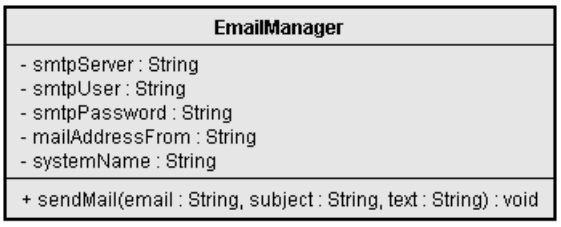

Abbildung 7.9.: Klassendiagramm EmailManager

Attribute

Name	Typ	Beschreibung
smtpServer	String	Adresse des SMTP-Servers
smtpUser	String	Benutzername für den SMTP Zugang
smtpPassword	String	Passwort für den SMTP Zugang
mailAddressFrom	String	E-Mailadresse von der versendet wird
systemName	String	Angezeigter Name des Absenders

Funktionen

sendMail() : void
Sendet eine E-Mail an die angegebene E-Mailadresse

Zugriff auf das E-Mailsystem Der Zugriff auf das E-Mailsystem läuft vergleichbar wie der Zugriff auf die Datenbank ab. Die Klasse EmailManager wird hierbei durch die Klassen des Modells vom Controller abgekapselt. Die send-Methoden der Klassen des Modells rufen hierbei die entsprechende Funktion des E-MailManagers auf. Abbildung 7.10 zeigt dieses Vorgehen anhand eines Beispiels mithilfe eines Sequenzdiagramms.

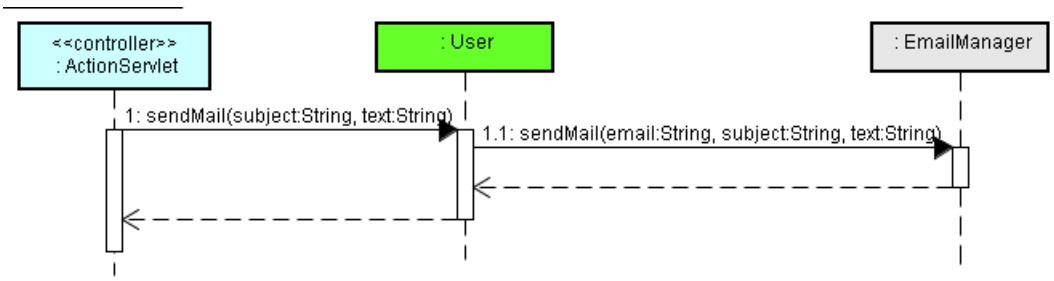

Abbildung 7.10.: Sequenzdiagramm E-Mails senden (Beispiel User.sendMail())

7.4.3. Servlets

Die Controller werden durch Java Servlets verwirklicht. Jede JSP erhält hierbei genau ein Servlet, welches den Zugriff auf die JSP's verwaltet und kontrolliert. Abbildung 7.11 gibt einen Überblick über die verwendeten Servlets.

7.4.3.1. *ActionServlet*

Die abstrakte Klasse ActionServlet dient als Basisklasse für jedes folgende Servlet. Sie enthält Funktionen und Attribute, die in jedem Servlet vorkommen und füllt diese beim Aufruf von doPost() oder doGet().

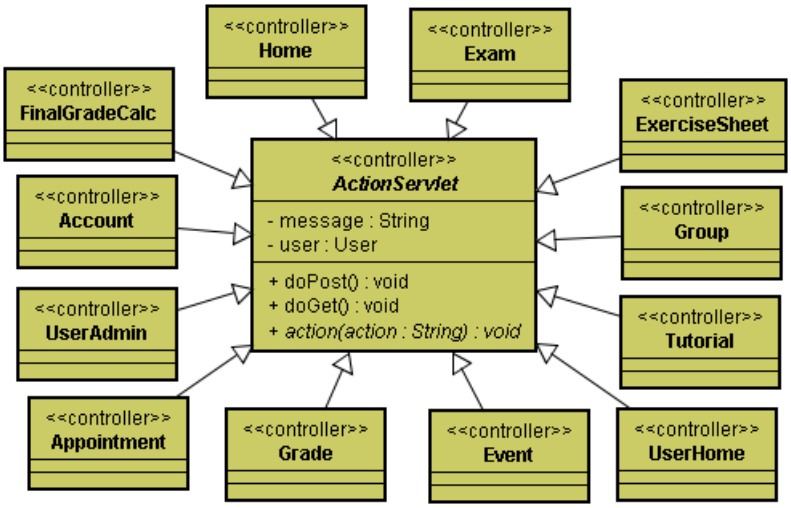

Abbildung 7.11.: Klassendiagramm Servlets

action(action : String) : void
Diese Funktion erhält einen String als Parameter, der die auszuführende Aktion beschreibt. Diese wird per HTTP-Parameter an das Servlet beim Auswählen einer Funktion im View übergeben. Diese Funktion muss von jedem Servlet implementiert werden. Sie sorgt dafür dass eine Funktion (im Folgenden als Action bezeichnet) im Servlet aufgerufen wird.

7.4.3.2. Home

Zuständigkeit (View): Das Servlet Home ist für home.jsp zuständig.

Actions:

Name	Beschreibung
home()	Bereitet Daten für home.jsp vor und leitet dorthin weiter.

7.4.3.3. Account

Zuständigkeit (View): Das Servlet Account ist für login.jsp, register.jsp, passwordrecovery.jsp und usersettings.jsp zuständig.

Actions:

Name	Beschreibung
showlogin()	Leitet zu login.jsp weiter wenn der Benutzer noch nicht angemeldet ist.
submitlogin()	Liest Eingaben in login.jsp aus und validiert diese. Sollte dies erfolgreich sein, wird der Benutzer angemeldet.
logout()	Löscht die aktuelle Session. Der Benutzer wird hierdurch abgemeldet.
showregister()	Zeigt register.jsp an, wenn der Benutzer noch nicht angemeldet ist.
submitregister()	Validiert die in register.jsp eingegebenen Daten und registriert der Benutzer, wenn die Daten korrekt sind. Der Benutzer erhält eine E-Mail mit einem Link mit dem er seinen Account aktivieren kann.
showrecoverpassword()	Leitet zu recoverpassword.jsp weiter wenn der Benutzer noch nicht angemeldet ist.
submitrecoverpassword()	Wenn eine vorhandene E-Mailadresse eingegeben wurde, wird ein neues Passwort generiert und dem Benutzer zugeschickt.
showusersettings()	Ist der Benutzer eingeloggt zeigt diese Action mithilfe von usersettings.jsp seine aktuellen Daten an.
submitusersettings()	Validiert die Eingaben in usersettings.jsp und ändert im Erfolgsfall die Daten in der Datenbank.

7.4.3.4. UserAdmin

Zuständigkeit (View): Das Servlet UserAdmin ist für useradministration.jsp zuständig.

Actions:

Name	Beschreibung
show()	Die Action leitet den Administrator zu useradministration.jsp um und zeigt dort eine Liste aller Benutzer des Systems an.
delete()	Löscht einen Benutzer.
submit()	Ändert die Daten eines Benutzers.

Alle Actions sind nur für Administratoren zugänglich.

7.4.3.5. Event

Zuständigkeit (View): Das Servlet Event ist für event.jsp und eventlist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt eine Veranstaltung, welche dann mithilfe von event.jsp angezeigt wird.
change()	Ist der Benutzer ein Administrator oder ein Dozent der Veranstaltung, so validiert diese Action eingegebene Datenänderungen und ändert im Erfolgsfall die Veranstaltung.
showlist()	Lädt alle Veranstaltungen und zeigt diese mit eventlist.jsp an.
add()	Ist der Benutzer ein Dozent oder ein Administrator, so erstellt diese Action aus eingegebenen Daten eine neue Veranstaltung und speichert sie.
delete()	Ist der Benutzer ein Administrator so löscht diese Action die ausgewählte Veranstaltung. Dozenten dürfen keine Veranstaltungen löschen, weil hierdurch auch alle Noten gelöscht werden.
addacademic()	Der Besitzer der Veranstaltung kann über diese Action weitere Teilnehmer der Veranstaltung zu eingeschränkten Dozenten machen.
deleteacademic()	Der Besitzer der Veranstaltung kann über diese Action eingeschränkte Dozenten entlassen.
addstudent()	Ein Benutzer, der noch nicht zur Veranstaltung angemeldet ist, kann sich durch diese Action anmelden.
deletestudent()	Ein Benutzer, der zur Veranstaltung angemeldet ist, kann sich durch diese Action abmelden.

7.4.3.6. Tutorial

Zuständigkeit (View): Das Servlet Tutorial ist für tutorial.jsp und tutoriallist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt ein Tutorium, welches dann mithilfe von tutorial.jsp angezeigt wird.
change()	Ist der Benutzer ein Administrator, Dozent der Veranstaltung oder der Tutor des Tutoriums, so validiert diese Action eingegebene Datenänderungen und ändert im Erfolgsfall das Tutorium.
showlist()	Lädt alle Tutorien der Veranstaltung und zeigt diese mit tutoriallist.jsp an.
add()	Ist der Benutzer ein Dozent oder ein Administrator, so erstellt diese Action aus eingegebenen Daten ein neues Tutorium und speichert es.
delete()	Ist der Benutzer ein Dozent oder ein Administrator so löscht diese Action das ausgewählte Tutorium. Noten der Studenten bleiben erhalten.
addtutor()	Über diese Action kann ein Dozent der Veranstaltung einen Tutor zum Tutorium hinzufügen.
deletetutor()	Hierüber kann ein Dozent der Veranstaltung einen Tutor aus dem Tutorium entfernen.
addstudent()	Ein Benutzer, der noch nicht zu einem Tutorium angemeldet ist, kann sich durch diese Action anmelden. Funktioniert nur wenn kein Wahlverfahren eingestellt wurde.
deletestudent()	Ein Benutzer, der zum Tutorium angemeldet ist, kann sich durch diese Action abmelden. Funktioniert nur wenn kein Wahlverfahren eingestellt wurde.

7.4.3.7. Group

Zuständigkeit (View): Das Servlet Group ist für group.jsp und grouplist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt eine Übungsgruppe, welches dann mithilfe von group.jsp angezeigt wird.
change()	Ist der Benutzer ein Administrator, Dozent der Veranstaltung oder der Tutor des Tutoriums, so validiert diese Action eingegebene Datenänderungen und ändert im Erfolgsfall die Übungsgruppe.
showlist()	Lädt alle Übungsgruppen des Tutoriums und zeigt diese mit grouplist.jsp an.
add()	Ist der Benutzer ein Dozent, der Tutor des Tutoriums oder ein Administrator, so erstellt diese Action aus eingegebenen Daten eine neue Übungsgruppe und speichert sie.
delete()	Ist der Benutzer ein Dozent, der Tutor des Tutoriums oder ein Administrator so löscht diese Action die ausgewählte Übungsgruppe.
addstudent()	Ein Benutzer, der noch nicht zu einer Übungsgruppe angemeldet ist, kann sich durch diese Action anmelden. Funktioniert nur wenn kein Wahlverfahren eingestellt wurde.
deletestudent()	Ein Benutzer, der zu einer Übungsgruppe angemeldet ist, kann sich durch diese Action abmelden. Funktioniert nur wenn kein Wahlverfahren eingestellt wurde.

7.4.3.8. Appointment

Zuständigkeit (View): Das Servlet Appointment ist für appointment.jsp und appointmentlist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt einen Termin, welcher dann mithilfe von appointment.jsp angezeigt wird.
change()	Ist der Benutzer ein Administrator oder Dozent der Veranstaltung, so validiert diese Action eingegebene Datenänderungen und ändert im Erfolgsfall den Termin.
showlist()	Lädt alle Termine des Tutoriums oder der Veranstaltung und zeigt diese mit appointmentlist.jsp an.
add()	Ist der Benutzer ein Dozent der Veranstaltung oder ein Administrator, so erstellt diese Action aus eingegebenen Daten einen neuen Termin und speichert ihn.
delete()	Ist der Benutzer ein Dozent der Veranstaltung oder ein Administrator so löscht diese Action den ausgewählten Termin.

7.4.3.9. Exam

Zuständigkeit (View): Das Servlet Exam ist für exam.jsp und examlist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt eine Prüfung, welche dann mithilfe von exam.jsp angezeigt wird.
change()	Ist der Benutzer ein Administrator oder Dozent der Veranstaltung, so validiert diese Action eingegebene Datenänderungen und ändert im Erfolgsfall die Prüfung.
showlist()	Lädt alle Prüfungen der Veranstaltung und zeigt diese mit examlist.jsp an.
add()	Ist der Benutzer ein Dozent der Veranstaltung oder ein Administrator, so erstellt diese Action aus eingegebenen Daten eine neue Prüfung und speichert sie.
delete()	Ist der Benutzer ein Dozent der Veranstaltung oder ein Administrator so löscht diese Action die ausgewählte Prüfung.

7.4.3.10. ExerciseSheet

Zuständigkeit (View): Das Servlet ExerciseSheet ist für exercisesheet.jsp und exercisesheetlist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt einen Übungszettel, welcher dann mithilfe von exercisesheet.jsp angezeigt wird.
change()	Ist der Benutzer ein Administrator oder Dozent der Veranstaltung, so validiert diese Action eingegebene Datenänderungen und ändert im Erfolgsfall den Übungszettel.
showlist()	Lädt alle Übungszettel der Veranstaltung und zeigt diese mit exercisesheetlist.jsp an.
add()	Ist der Benutzer ein Dozent der Veranstaltung oder ein Administrator, so erstellt diese Action aus eingegebenen Daten einen neuen Übungszettel und speichert ihn.
delete()	Ist der Benutzer ein Dozent der Veranstaltung oder ein Administrator, so löscht diese Action den ausgewählten Übungszettel.

7.4.3.11. FinalGradeCalc

Zuständigkeit (View): Das Servlet FinalGradeCalc ist für finalgradecalculation.jsp zuständig.

Actions:

Name	Beschreibung
show()	Lädt die aktuelle Berechnung der Endnote und zeigt diese mit finalgradecalculation.jsp an.
submit()	Validiert die eingegebene Berechnungsvorschrift von finalgradecalculation.jsp und speichert diese bei Erfolg ab.

Alle Actions können nur von Dozenten der Veranstaltung und Administratoren verwendet werden.

7.4.3.12. Grade

Zuständigkeit (View): Das Servlet Grade ist für studentgradelist.jsp zuständig.

Actions:

Name	Beschreibung
show()	Zeigt die Studentennoten einer Veranstaltung oder eines Tutoriums an. Nur Tutoren oder Dozenten können diese Action benutze. Tutoren sehen nur die Noten der Studenten seines Tutoriums.
submit()	Validiert Änderungen an den Noten und speichert diese im Erfolgsfall. Kann ebenfalls nur durch Tutoren und Dozenten benutzt werden.

7.4.3.13. UserHome

Zuständigkeit (View): Das Servlet UserHome ist für userhome.jsp zuständig.

Actions:

Name	Beschreibung
show()	Bereitet benutzerspezifische Daten vor, die in userhome.jsp angezeigt werden.

7.5. Datenbankentwurf

7.5.1. Konzeptioneller Entwurf

Die im Modell festgelegten Daten, sollen persistent gespeichert werden. Verwendet wird hierfür eine relationale Datenbank des PostgreSQL Datenbankmanagementsystems. Im Folgenden wird ein konzeptioneller Entwurf des Datenbankmodells mithilfe eines ER-Diagramms durchgeführt. Als Strategie wurde das Bottom-Up-Verfahren angewendet, indem erst die einzelnen Packages model.user, model.event und model.grade unabhängig voneinander modelliert und schließlich zusammengefügt wurden. In Anhang A findet man das resultierende ER-Diagramm, wobei Generalisierungen bereits umgewandelt wurden. Im Falle des Users ist eine überlappende Generalisierung vorhanden. Ein User kann theoretisch sowohl Student, als auch Dozent, Tutor und Administrator sein. Daher werden hier die drei booleschen Werte „isAdmin“, „isAcademic“ und „isTutor“ eingeführt. Im Falle der Zwischennoten reicht ein Attribut „type“ aus, welches festlegt welche Art von Zwischennote der Eintrag repräsentiert. Für die Umsetzung wird außerdem eine Enumeration im Klassenmodell hinzugefügt:

Name	Werte	Beschreibung
GradeType	Exam, ExerciseSheet, Exercise	Beschreibt um welche Art von Zwischennote es sich handelt

7.5.2. Resultierendes Relationenschema

Folgende Relationen resultieren aus dem ER-Diagramm (unterstrichene Attribute stellen den Primärschlüssel dar):

User

Attribut	Datentyp	Beschreibung
email	varchar(50)	E-Mail-Adresse
password	char(32)	Passwort MD5 verschlüsselt (32 Byte)
title	varchar(20)	Titel (z.B. Professor)
forename	varchar(20)	Vorname (not NULL)
surname	varchar(20)	Nachname (not NULL)
address	varchar(50)	komplette Anschrift
phone	varchar(20)	Telefonnummer
homepage	varchar(50)	Homepage
isActive	boolean	Account aktiviert?
isAdmin	boolean	Hat Administratorrechte?
isAcademic	boolean	Hat Dozentenrechte?
isTutor	boolean	Ist Tutor?
wantsEventNotifications	boolean	Möchte Benachrichtigungen bei Veranstaltungsveränderungen (Default true)
wantsGradeNotifications	boolean	Möchte Benachrichtigungen bei Notenänderung (Default true)
wantsRightNotifications	boolean	Möchte Benachrichtigungen bei Änderung der Rechte (Default true)
wantsGroupNotifications	boolean	Möchte Benachrichtigungen bei Gruppenänderungen (Default true)

Event

Attribut	Datentyp	Beschreibung
name	varchar(50)	Name der Veranstaltung
semester	varchar(8)	Semester der Veranstaltung
ownerEmail	varchar(50)	Besitzender Dozent der Veranstaltung (Fremdschlüssel User)
isOpen	boolean	Offen für Anmeldungen? (Default true)
password	char(32)	optionales Passwort
finalGradeCalculationString	varchar(100)	Zeichenfolge beschreibt die Berechnung der Endnote innerhalb der Veranstaltung

Tutorial

Attribut	Datentyp	Beschreibung
name	varchar(50)	Name des Tutoriums
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Event)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Event)
isOpen	boolean	Offen für Anmeldungen? (Default true)
maxParticipants	integer	Maximale Anzahl an Teilnehmern

Group

Attribut	Datentyp	Beschreibung
name	varchar(50)	Name der Übungsgruppe
tutorialName	varchar(50)	Name des übergeordneten Tutoriums (Fremdschlüssel Tutorial)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Tutorial)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Tutorial)

Appointment

Attribut	Datentyp	Beschreibung
id	SERIAL	ID zur Identifizierung des Termins
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Event)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Event)
tutorialName	varchar(50)	Name des übergeordneten Tutoriums (Fremdschlüssel Tutorial) Wenn ungleich NULL, dann gilt der Termin für das Tutorium, sonst für die Veranstaltung.
frequency	integer	Zahlenwert der Enumeration Frequency (Abschnitt 7.2.4.4)
day	integer	Tag
month	integer	Monat als Zahl (0..11)
year	integer	Jahr (vierstellig)
hour	integer	Stunde (Uhrzeit Begin)
minute	integer	Minute (Uhrzeit Begin)
toHour	integer	Stunde (Uhrzeit Ende)
toMinute	integer	Minute (Uhrzeit Ende)
weekday	integer	Zahlenwert der Enumeration Weekday (Abschnitt 7.2.4.4)
location	varchar(50)	Ort oder Raumangabe
description	varchar(50)	weitere Beschreibung des Termins

Registration

Attribut	Datentyp	Beschreibung
matriculationNumber	integer	Matrikelnummer
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Event)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Event)
wasAttendant	boolean	War der Angemeldete Student bei der Prüfung anwesend? (Default false)
finalGrade	float	Endnote
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)

HalfGrade

Attribut	Datentyp	Beschreibung
name	varchar(50)	Name der Zwischennote
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Event)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Event)
availableScore	numeric(4,1)	Mögliche Punktzahl
type	integer	Zahlenwert der Enumeration GradeType (Abschnitt 7.5.1)

Exercise

Attribut	Datentyp	Beschreibung
number	integer	Nummer der Aufgabe
halfGradeName	varchar(50)	Name der übergeordneten Zwischennote (Fremdschlüssel HalfGrade)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel HalfGrade)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel HalfGrade)
availableScore	numeric(4,1)	Mögliche Punktzahl

has_result_in_halfgrade

Attribut	Datentyp	Beschreibung
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)
halfGradeName	varchar(50)	Name der Zwischennote (Fremdschlüssel HalfGrade)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel HalfGrade)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel HalfGrade)
result	numeric(4,1)	Ergebnis des Studenten

has_result_in_exercise

Attribut	Datentyp	Beschreibung
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)
number	integer	Nummer der Aufgabe (Fremdschlüssel Exercise)
halfGradeName	varchar(50)	Name der Zwischennote (Fremdschlüssel Exercise)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Exercise)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Exercise)
result	numeric(4,1)	Ergebnis des Studenten

event_has_students

Attribut	Datentyp	Beschreibung
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Event)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Event)
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)

event_has_academics

Attribut	Datentyp	Beschreibung
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Event)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Event)
userEmail	varchar(50)	E-Mailadresse des Dozenten (Fremdschlüssel User)

tutorial_has_students

Attribut	Datentyp	Beschreibung
tutorialName	varchar(50)	Name des Tutoriums (Fremdschlüssel Tutorial)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Tutorial)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Tutorial)
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)

tutorial_has_tutors

Attribut	Datentyp	Beschreibung
tutorialName	varchar(50)	Name des Tutoriums (Fremdschlüssel Tutorial)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Tutorial)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Tutorial)
userEmail	varchar(50)	E-Mailadresse des Tutors (Fremdschlüssel User)

group_has_students

Attribut	Datentyp	Beschreibung
groupName	varchar(50)	Name der Übungsgruppe (Fremdschlüssel Group)
tutorialName	varchar(50)	Name des Tutoriums (Fremdschlüssel Group)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel Group)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel Group)
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)

exams

Attribut	Datentyp	Beschreibung
halfGradeName	varchar(50)	Name der Zwischennote die dem Examen entspricht (Fremdschlüssel HalfGrade)
eventName	varchar(50)	Name der übergeordneten Veranstaltung (Fremdschlüssel HalfGrade)
eventSemester	varchar(8)	Semester der übergeordneten Veranstaltung (Fremdschlüssel HalfGrade)
appointmentId	integer	ID des Termins der Prüfung (Fremdschlüssel Appointment)
signinPeriodId	integer	ID des Termins, der den Anmeldezeitraum festlegt (Fremdschlüssel Appointment)

Das Relationsschema ist bereits in 3. Normalform (siehe auch [ENS03]) und muss daher nicht weiter normalisiert werden.

user_activation

Diese Relation hält die Zeichenfolge zur Accountaktivierung. Einträge werden nach der Aktivierung gelöscht.

Attribut	Datentyp	Beschreibung
userEmail	varchar(50)	E-Mailadresse des Studenten (Fremdschlüssel User)
activation	char(32)	Zeichenfolge zur Accountaktivierung
createTime	timestamp	Zeitpunkt der Registrierung

Systemlog

Zusätzlich wird noch folgende Relation eingefügt, die wichtige Systemereignisse mitloggen soll. Sie ist im ER-Diagramm nicht aufgeführt, da Fremdschlüsselbeziehungen hier nicht übernommen werden. Das Löschen von Fremdschlüsseln soll nicht zur Löschung eines Logeintrags führen.

Attribut	Datentyp	Beschreibung
id	integer	ID des Eintrags
emailUser	varchar(50)	Benutzer der das Ereignis ausgelöst hat
eventName	varchar(50)	Name der Veranstaltung
eventSemester	varchar(8)	Semester der Veranstaltung
logEventCode	integer	Beschreibt den Typ des Systemereignisses (siehe nächster Paragraph)
description	varchar(200)	Beschreibung des Ereignisses
logTimestamp	timestamp	Zeitpunkt des Ereignisses

Systemereignisse

Die folgenden Systemereignisse sollen mitgeloggt werden (Zahl entspricht logEventCode):

- 0 : Benutzer hinzugefügt
- 1 : Benutzer gelöscht
- 2 : Benutzerrecht geändert
- 3 : Veranstaltung hinzugefügt
- 4 : Veranstaltung geändert
- 5 : Veranstaltung gelöscht
- 6 : Tutorium hinzugefügt
- 7 : Tutorium geändert
- 8 : Tutorium gelöscht
- 9 : Übungsgruppe hinzugefügt
- 10: Übungsgruppe geändert

- 11: Übungsgruppe gelöscht
- 12: Note hinzugefügt
- 13: Note geändert
- 14: Note gelöscht
- 15: Notenberechnung hinzugefügt
- 16: Notenberechnung geändert
- 17: Notenberechnung gelöscht
- 18: Zwischennote hinzugefügt
- 19: Zwischennote geändert
- 20: Zwischennote gelöscht
- 21: Endnote eingetragen

7.5.3. Datenbankzugriff

Wie bereits in Abschnitt 6.2.2 erwähnt wurde, wird auf die Datenbank mithilfe eines JDBC Typ-4 Treibers zugegriffen. Da der Verbindungsaufbau zur Datenbank als eines der teuersten Prozeduren gilt, wird ein Connection-Pooling verwendet, sodass mehrere Verbindungen zur Datenbank aufrecht erhalten werden. Voneinander abhängige Querys und Updates werden innerhalb von Transaktionen vorgenommen. Im Fehlerfall kann so ein „Rollback“ stattfinden, welches alle Änderungen der Transaktion rückgängig macht. Somit wird verhindert, dass die Datenbank in einen inkonsistenten Zustand gerät.

7.5.4. Verwendete Bibliotheken

JavaServer Pages Standard Tag Library (JSTL): Die JSTL wird verwendet um JSPs übersichtlicher und strukturierter gestalten zu können. Durch diese Bibliothek kann es vermieden werden normalen Javacode in JSPs zu verwenden. Die Bibliothek bietet hierfür eine ganze Reihe von nützlichen Tags. Mehr Informationen findet man unter `http://java.sun.com/products/jsp/jstl/`.

Java Secure Socket Extensions (JSSE): Die JSSE wird verwendet um eine sichere HTTPS Verbindung zwischen Browser und Server zu verwirklichen. Die Bibliothek ist ab Java JRE Version 5 in jedem JRE enthalten. Mehr Informationen über diese Bibliothek findet man unter `http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html`

Java Server Pages 2.1 (JSP) Die Java Server Pages werden verwendet um dynamischen Webseiten zu erzeugen. Hierfür wurde Version 2.1 benutzt. Mehr Informationen über JSPs findet man unter `http://java.sun.com/products/jsp/`

Java Database Connectivity (JDBC): Der JDBC Typ-4 Treiber der Firma PostgreSQL (Version 8.3-604) wurde verwendet um die Verbindung zur Datenbank über Java herstellen zu können. JDBC bietet hierfür eine plattformunabhängige, einheitliche Programmierschnittstelle für Java-Anwendungen. Mehr Informationen über JDBC findet man unter `http://java.sun.com/javase/technologies/database/`

JavaMail 1.4.2: JavaMail ist eine Bibliothek für die Kommunikation mit E-Mailservern. Sie wurde verwendet um den Benachrichtigungsservice im System zu verwirklichen. Unter `http://java.sun.com/products/javamail/` findet man weitere Informationen über JavaMail

Javascript Framework jQuery 1.3.1: jQuery ist das gewählte AJAX Framework für diese Projekt. Es bietet viele Funktionalitäten und viele PlugIns zur Erweiterung an. Mehr Informationen findet man unter `http://jquery.com/`

8. Zusammenfassung und Ausblick

8.1. Ergebnisse

Das Ergebnis dieser Arbeit ist ein Informationssystem um eine Hochschulveranstaltung zu planen und zu verwalten. Hierbei können Noten, Studenten, Tutoren, Dozenten, Veranstaltungen, Tutorien, Übungsgruppen, Übungszettel, Prüfungen und Endnotenberechnungen verwaltet werden. Außerdem bietet es durch sein E-Mailbenachrichtigungssystem einen Service stets über Änderungen, die einen selbst betreffen, informiert zu sein. Durch die modulare Entwicklung eignet sich das Projekt sehr gut für das Refactoring und kann somit leicht erweitert oder geändert werden. Die folgende Tabelle zeigt eine Zusammenfassung der gestellten Anforderungen und dessen Umsetzung.

Anwendungsfall	**Name**	**Umsetzung**	**Abschnitt**
GA-01	Veranstaltungsdetails anzeigen	Klasse Event, Servlet Event, event.jsp	7.2.4.2
GA-02	Anmelden	Klasse User, Servlet Account, login.jsp	7.2.3.1
GA-03	Registrieren	Klasse User, Servlet Account, register.jsp	7.2.3.1
GA-04	Neues Passwort zuschicken	Klasse User, Servlet Account, passwordrecovery.jsp	7.2.3.1
ST-01	Persönliche Daten ändern	Klasse User, Servlet Account, usersettings.jsp	7.2.3.1
ST-02	Benachrichtigungsoptionen einstellen	Klasse E-MailManager, Klasse User, Servlet Account, usersettings.jsp	7.2.3.1 7.4.2.2
ST-03	Zu Veranstaltung an-/abmelden	Klasse Event, Servlet Event, event.jsp und eventlist.jsp	7.2.4.2
ST-04	Eigene Noten einsehen	Klasse Result, Servlet Grade, userhome.jsp und studentgradelist.jsp	7.2.5.6
TU-01	Allgemeine Daten ändern	Klasse Tutorial, Servlet Tutorial, tutorial.jsp	7.2.4.3
TU-02	Übungsgruppen einteilen	Klasse Group, Servlet Group, groupchoise.jsp, group.jsp, grouplist.jsp	7.2.4.5
TU-03	Noten verwalten	Klasse Result, Klasse HalfGrade, Servlet Grade, studentgradelist.jsp markieren von Änderungen noch nicht implementiert	7.2.5.6 7.2.5.1
DO-01	Tutoren verwalten	Klasse Tutorial, Klasse Tutor, Servlet Tutorial, tutorial.jsp	7.2.4.3 7.2.3.3

DO-02	Tutorien verwalten	Klasse Tutorial, Servlet Tutorial, tutorial.jsp und tutoriallist.jsp	7.2.4.3
DO-03	Noten verwalten	Klasse Result, Klasse HalfGrade, Servlet Grade, studentgradelist.jsp	7.2.5.6 7.2.5.1
DO-04	Studenten verwalten	Klasse Event, Klasse Student, Servlet Event, event.jsp	7.2.4.2 7.2.3.4
DO-05	Zugriffsrechte verwalten	Nur teilweise umgesetzt durch das Hinzufügen von eingeschränkten Dozenten. Klasse Event, Klasse User, Servlet Event, event.jsp	7.2.4.2 7.2.3.1
AD-01	Benutzer verwalten	Klasse User, Servlet UserAdmin, useradministration.jsp	7.2.3.1
AD-02	Dozenten ernennen	Klasse User, Klasse Academic, Servlet UserAdmin, useradministration.jsp	7.2.3.1 7.2.3.2
AD-03	Veranstaltungen löschen	Klasse Event, Servlet Event, eventlist.jsp	7.2.4.2
	Sichere Verbindungen	Verwendung von HTTPS	6.2.1

Servlets können in Abschnitt 7.4.3 und JSPs können in Abschnitt 7.3.1 nachgeschlagen werden. Die Verweise zu den verwendeten Klassen des Modells sind in der Spalte „Verweise“ zu finden.

8.2. Alternativen

Benutzt man aktuelle Suchmaschinen um nach Verwaltungsprogrammen für Schulnoten zu suchen, so erhält man eine riesige Anzahl an Treffern. Im Folgenden sollen der Vollständigkeit halber zwei Alternativen zu diesem Projekt vorgestellt werden.

Online Grades: Online Grades ist eine Webanwendung für Schulen, welche ebenfalls zur Verwaltung von Noten verwendet werden kann. Die Organisation der Lehreinheiten wird hierbei auf Klassen beschränkt (z.B. Englisch Klasse). Noten werden einfach eingetragen. Schüler und ihre Eltern können die Noten des Schülers einsehen. Für eine Universitätsveranstaltung wäre dies aber durch die starke Orientierung an eine Schulveranstaltung nicht zu gebrauchen. Unter `http://www.onlinegrades.org` erhält man weitere Informationen, sowie eine Demo der Software. OpenGrade (`http://www.lightandmatter.com/ogr/ogr.html`) ist ein Desktoptool, welches ebenfalls die Möglichkeit hat auf das System von Online Grades zuzugreifen.

Stud.IP: Das Stud.IP ist die E-Learning Plattform der Universität Oldenburg und bietet sehr ähnliche Funktionalitäten, wie dieses Projekt. Es bietet sehr viele Funktionen, wie z.B. Dateiuploads und diverse Communityfunktionen. Unter `http://elearning.uni-oldenburg.de` ist es erreichbar.

8.3. Ausblick

Da bei der Entwicklung des Systems auf Erweiterbarkeit geachtet wurde, könnte man noch einige Features hinzufügen. Ein Offlinetool, welches die Bearbeitung von Noten ohne Internetverbindung ermöglicht, wäre neben einem Textparser für Noten sinnvoll. Vor allem wäre eine für PDAs optimierte Version wünschenswert. Auch die Seite selbst könnte für die Benutzung von internetfähigen Handys optimiert werden, da diese nur sehr eingeschränkte Javascript und AJAX Fähigkeiten besitzen. Möglich wären auch verschiedene Foren, die die Möglichkeit bieten könnten, über Veranstaltungen und andere Dinge zu diskutieren. Ein Gästebuch auf seiner eigenen Benutzerseite wäre ebenfalls eine Option.
Veranstaltungen könnten noch über eine Dateidatenbank erweitert werden um Vorlesungsfolien und Übungszettel hochladen zu können. Das Rechtesystem könnte verfeinert werden, sodass Tutoren weitere Änderungsrechte an einer Veranstaltung erhalten können, ohne zum Dozent ernannt werden zu müssen.
Sollte es zu einer übermäßigen Belastung der Datenbank kommen, so könnte man einmal geladene Daten an einer zentralen Position im Speicher (z.B. durch ein Singleton-Objekt) halten und Datenanfragen erst hier vornehmen, bevor ein Datenbankzugriff erfolgt.
Aktuelle Javascript Frameworks können dynamisch Diagramme erzeugen. Eine weitere Möglichkeit der Erweiterung des Systems wäre dies zu Nutzen um Notenverteilungen grafisch aufzubereiten und anzuzeigen.

A. ER-Diagramm

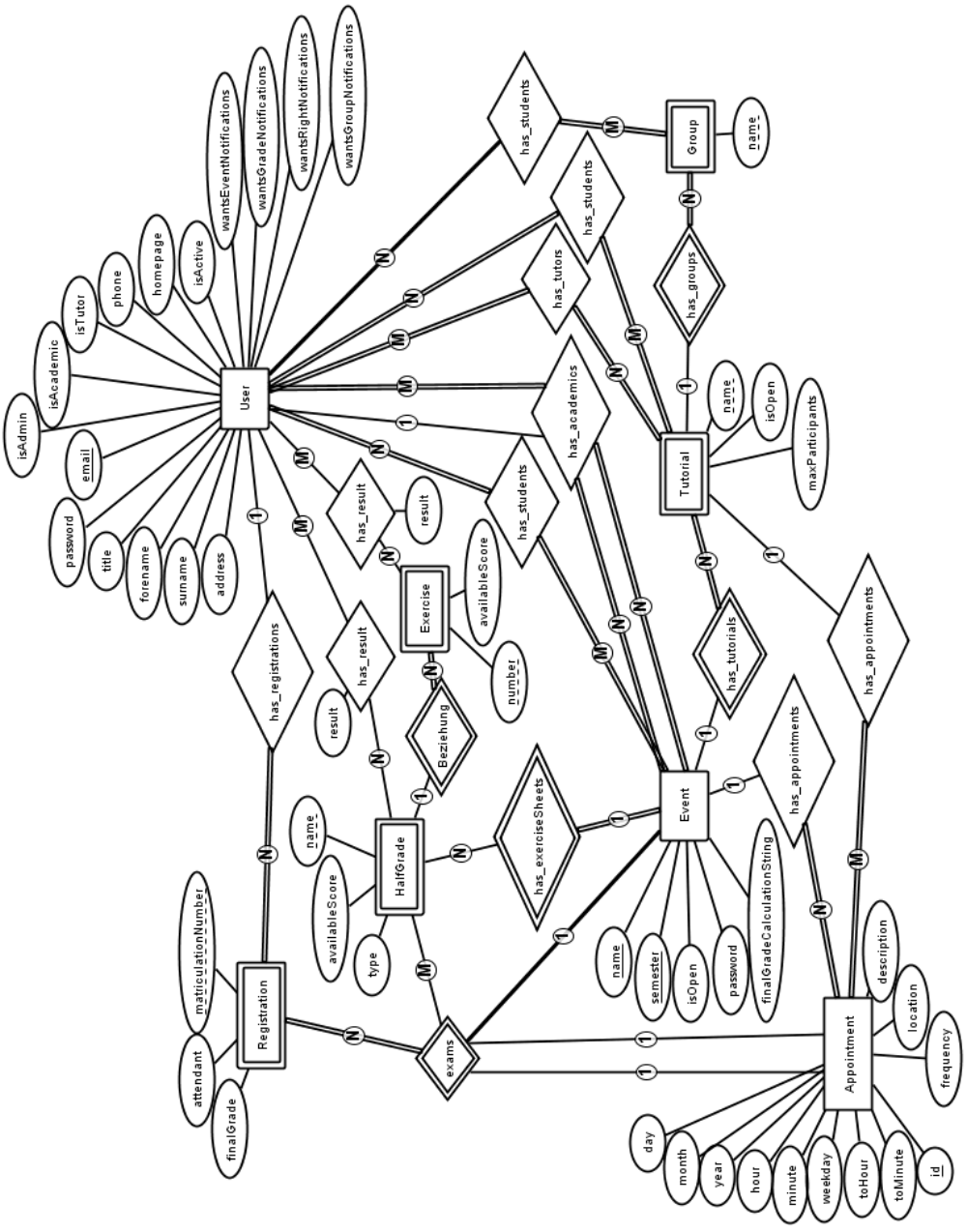

Abbildung A.1.: ER-Diagramm des Datenbankentwurfs

Literaturverzeichnis

[ENS03] ELMASRI, Ramez ; NAVATHE, Shamkant B. ; SHAFIR, Angelika: *Grundlagen von Datenbanksystemen*. Pearson Studium, 2003. – ISBN 3–8273–7021–3

[Fou] FOUNDATION, Apache S.: *Apache Tomcat 6.0 - SSL Configuration HOW-TO*. `http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html`: Apache Software Foundation

[Gro99] GROUP, Network W.: *Hypertext Transfer Protocol – HTTP/1.1*. `http://tools.ietf.org/html/rfc2616`: Network Working Group, Juni 1999

[Gro00] GROUP, Network W.: *HTTP Over TLS*. `http://tools.ietf.org/html/rfc2818`: Network Working Group, Mai 2000

[Gro08] GROUP, Network W.: *The Transport Layer Security (TLS) Protocol*. `http://tools.ietf.org/html/rfc5246`: Network Working Group, August 2008

[SDNa] SDN: *Code Conventions for the Java Programming Language*. `http://java.sun.com/docs/codeconv/`: Sun Developer Network (SDN)

[SDNb] SDN: *JavaBeans Spec*. `http://java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html`: Sun Developer Network (SDN)

[Som07] SOMMERVILLE, Ian: *Software Engineering*. Pearson Studium, 2007. – ISBN 3–8273–7257–7

[Wik09] WIKIPEDIA: *Erweiterte Backus-Naur-Form*. `http://de.wikipedia.org/wiki/Erweiterte_Backus-Naur-Form`: Wikimedia, Februar 2009