

Datenbankpraktikum 27.07. - 14.08.2009

Oliver Vornberger, Nicolas Neubauer, Marcel Trame
University of Osnabrück

Inhaltsverzeichnis

1. Konzeptuelle Modellierung	2
1.1 Rollenbeschreibung.....	2
1.2 Datenbankmodellierung.....	3
1.3 Oberflächenanforderung.....	6
1.4 Modularisierung.....	7
2. Planung	9
2.1 Layout mit CSS und jQuery, Autocomplete-Funktion und Veranstaltung-Formular.....	9
2.2 Template-Engines.....	16
2.3 PlotKit.....	19
3. Login, Menü & Statistiken	27
3.1 Grundgerüst und Login.....	27
3.2 Generierung des Menüs.....	30
3.3 Statistiken.....	33
4. Studenten- und Gruppenverwaltung	38
5. Prüfungsverwaltung	49
6. Klausurpunkte eintragen	58
7. Veranstaltungsverwaltung	62
8. Seminarnote	68
9. Punkte für Übungsblätter	70
10. Personen- und Angestelltenverwaltung	73

1. Konzeptuelle Modellierung

von D. Abraham, A. Brankova, S. Heider, D. Künne, K. Roehr, A. Siemer, K. Sperber

Die konzeptuelle Modellierung befasst sich mit der Ausarbeitung der Konzepte, der verantwortlichen Parameter und des gesamten Grundgerüsts. Sie umfasst die Modellierung der Datenbank und der ER-Sicht und einer groben Übersicht über die Anforderung an die Benutzeroberfläche sowie die Rollen- und Rechteverteilung innerhalb der Programmhierarchie.

1.1 Rollenbeschreibung

von D. Abraham, A. Brankova, S. Heider, D. Künne, K. Roehr, A. Siemer, K. Sperber

Um mit dem System arbeiten zu können, muss jeder Benutzer über einen Login verfügen, mit dem er sich anmelden kann und anhand dessen ihm eine eindeutige Rolle in den Veranstaltungen zugewiesen werden kann. So soll es zum Beispiel möglich sein, dass ein Student in einer Vorlesung nur Student, in einer anderen aber als Tutor tätig ist. Somit verfügt er abhängig von der gewählten Rolle über verschiedene Berechtigungen.

Administrator

- darf Dozenten / Übungsleiter anlegen und für Veranstaltungen freischalten
- darf Studenten zu Tutoren machen
- darf Studenten anlegen und Studentendaten aus Stud.IP importieren
- dürfte bei Interesse auch Auswertungen lesen

Dozent / Übungsleiter

- darf andere Dozenten / Übungsleiter anlegen und für Veranstaltungen freischalten
- darf Studenten in seinen Veranstaltungen zu Tutoren machen
- darf Endnoten in seinen Veranstaltungen eintragen
- kann weitere Veranstaltungen erstellen
- darf Studenten anlegen und Studentendaten aus Stud.IP importieren
- kann sich Statistiken zu seinen Veranstaltungen anzeigen lassen
- kann einen abweichenden Notenspiegel für seine Klausuren einstellen
- darf Übungszettel mit Aufgaben und Punkten erstellen und ändern
- darf Zulassungseinstellungen für die Klausuren setzen
- verfügen zusätzlich noch über alle Eigenschaften eines Tutors

Tutor

- darf Testatgruppen anlegen / Studenten eintragen / Gruppen bearbeiten
- darf für seine Tutanden / Gruppen Punkte vergeben
 - beim Testatbetrieb nur für seine Gruppen
 - bei den Klausuren für alle Teilnehmer

- darf sich Statistiken anzeigen lassen
 - beim Testatbetrieb für seine eigenen Tutanten
 - bei den Klausuren für die gesamte Vorlesung

Student

- darf seine eigenen Statistiken anschauen
- seine persönlichen Daten validieren und ggf. ändern

1.2 Datenbankmodellierung

von D. Abraham, A. Brankova, S. Heider, D. Künne, K. Roehr, A. Siemer, K. Sperber

ursprüngliches ER-Diagramm

Für die Planung der Datenbank haben wir im ersten Schritt ein grobes ER-Diagramm erstellt, das alle benötigten Entitäten darstellt. Während der Programmierung des Web-Interfaces ist uns aber aufgefallen, dass an einigen Stellen noch Änderungen / Erweiterungen vorgenommen werden sollten. Diese haben dann zum endgültigen ER-Diagramm (siehe: Abbildung 2) geführt.

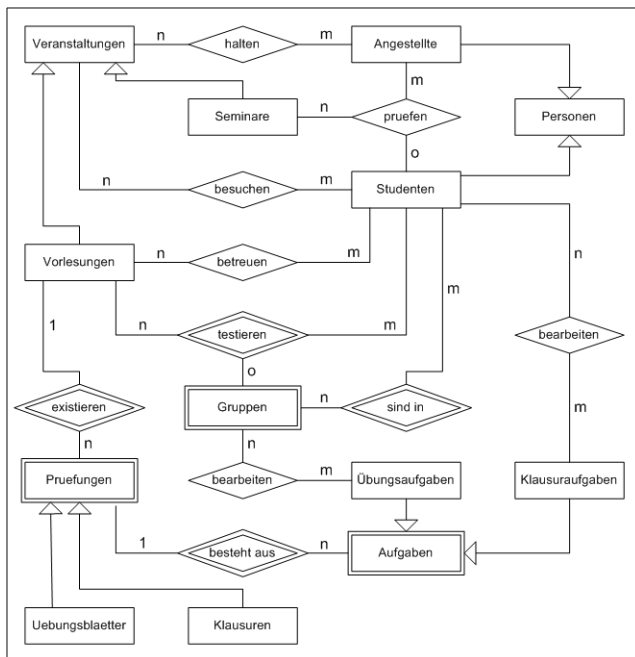


Abb. 1: ursprüngliches ER-Diagramm für die Testat- und Klausurverwaltung

ursprüngliches Schema der Datenbank

Veranstaltungen	:{[VerID : integer, Titel : string, Semestertyp : enum, Semesterjahr : integer maxStudenten : integer]}
Vorlesungen	:{[VerID : integer, proZettel : boolean, Prozentsatz : double, AnzFreiversuche : integer, Gruppengroesse : integer, Uebungsblaetter : integer]}
Seminare	:{[VerID : integer]}
Personen	:{[Login : string, Vorname : string, Name : string, Titel : string, E-Mail : string, Passwort : string]}
Angestellte	:{[Login : string, Rolle : enum]}
Studenten	:{[Login : string, Einschreibedatum : date, Studiengang : string, MatrNr(UNIQUE) : integer]}
Gruppen	:{[GrpID : integer, VerID : integer, TutorLog : string, Kuerzel : string]}
Pruefungen	:VIEW Klausuren + Uebungen + berechnete Pruefungsart
Klausuren	:{[VerID : integer, KlausurNr : integer, Datum : date, Aufgaben : integer]}
Uebungsblaetter	:{[VerID : integer, BlattNr : integer, Woche : integer, Aufgaben : integer]}
Aufgaben	:{[VerID : integer, PrID : integer, Art : enum, AufgNr : integer, maxPunkte : integer]}
Klausuraufgaben	:VIEW mit einem Filter auf Aufgaben
Uebungsaufgaben	:VIEW mit einem Filter auf Aufgaben
halten	:{[VerID : integer, Login : string]}
betreuen	:{[Login : string, VerID : integer]}
besuchen	:{[Login : string, VerID : integer]}
pruefen	:{[VerID : integer, StudLogin : string, AngLogin : string, Note : float]}
sindIn	:{[GrpID : integer, VerID : integer, Login : string, von : integer, bis : integer]}
bearbeitenUeb	:{[VerID : integer, GrpID : integer, PrID : integer, Art : enum, AufgNr : integer, Punkte : integer]}
bearbeitenKL	:{[VerID : integer, Login : string, PrID : integer, Art : enum, AufgNr : integer, Punkte : integer]}

Umsetzung auf dem MySQL-Server / Änderungen

von F.Hebestreit, P. Middendorf, D.Künne

Umsetzung auf dem Server

Beim Erstellen der Datenbank mussten neben dem Anlegen der Tabellen auch sämtliche Fremdschlüssel mit angelegt werden, damit die Abhängigkeiten zwischen den einzelnen Entitäten gegeben sind. So darf zum Beispiel keine Prüfung mehr existieren, wenn es die zugehörige Vorlesung nicht mehr gibt. Außerdem wurden einige Funktionen angelegt, die das Arbeiten wesentlich vereinfachen sollen.

```

1 CREATE FUNCTION naechsteGruppenNr (VID int) RETURNS int(11)
2   READS SQL DATA
3 BEGIN
4   SET @max = ( SELECT MAX( GrpID ) FROM Gruppen WHERE VerID = VID );
5   IF (@max is null) THEN
6     RETURN 0;
7   END IF;
8   RETURN @max +1;
9 END

```

überarbeitetes ER-Diagramm

Beim Erstellen der Datenbank und der ersten Models wurde festgestellt, dass die Entities Übungsaufgaben und Klausuraufgaben nie direkt angesprochen werden. Dementsprechend wurden die beiden Views aus dem Konzept herausgenommen. Ähnliches gilt für die Prüfungen. Hier wurde die Entity *Prüfungen* deswegenn als Tabelle und nicht als View umgesetzt.

Des Weiteren wurden die Entitäten *Notenschluessel* und *ECTS-Grades* eingeführt um datenbankseitig bereits die Zuordnung einer Note zu der erreichten Punktzahl vornehmen zu können. Diese Änderung geschah hauptsächlich aus Gründen der Performance, da eine Berechnung in der Datenbank deutlich schneller ist, als mit PHP.

Um die Statistiken korrekt umsetzen zu können und dabei zu berücksichtigen, dass ein Student in mehreren Studiengängen eingeschrieben sein kann, wurde noch die Entity *Studiengaenge* eingeführt. Außerdem muss noch eine Historie für alte Matrikelnummern geführt werden, da es Fälle gibt, in denen ein Student im Laufe seines Studiums mehrere Matrikelnummern hat.

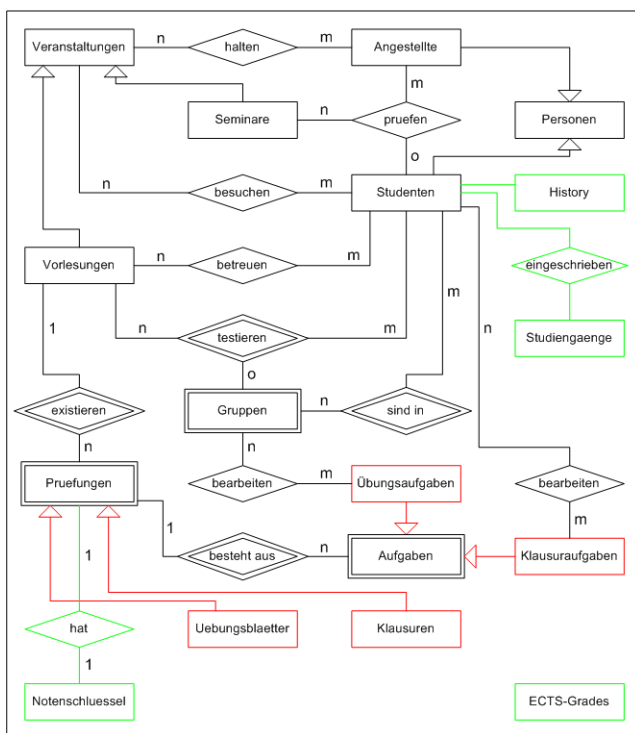


Abb. 2: überarbeitetes ER-Diagramm aufgrund von Anforderungsänderungen

neue Entitäten im Schema

Durch diese Änderungen musste auch das Schema erweitert werden und es kamen folgende neue Entitäten und Relationships hinzu.

History : {[Login : string, alteMatrNr : integer, von : date, bis : date]}

ECTS-Grade : {[Grade : varchar, Note : float]}

Notenschluessel: {[VerID : integer, PrID : integer, Typ: enum, Note: float, MaxPunkte : integer]}

Studiengaenge : {[StdgID : integer, Studiengang : string]}

eingeschrieben : {[Login : string, StdgID : integer]}

1.3 Oberflächenanforderung

von D. Abraham, A. Brankova, S. Heider, D. Künne, K. Roehr, A. Siemer, K. Sperber

Zur Definition der Anforderungen an die GUI haben wir die einzelnen Aufgaben grob in einzelne Themengebiete zusammengefasst.

Studentenverwaltung

- Studenten anlegen mit all ihren Daten
- Import von Studentendaten aus .csv-Datei (Daten aus Stud.IP)
- Studenten in eine Veranstaltung eintragen
- Daten einer Person modifizieren
 - History aufbauen für Änderungen der Matrikelnummer
- Student löschen

Tutorenverwaltung (als Teil der Veranstaltungsverwaltung)

- Studenten in dieser Veranstaltung zu Tutoren machen
- Tutoren löschen (als Student bleibt die gewählte Person erhalten)

Veranstaltungsverwaltung

- Übungsleiter und Tutoren anlegen und in die Veranstaltung eintragen
 - beim Anlegen neu erstellen, falls die Personen noch nicht in der Datenbank vorhanden ist
- Studenten importieren und manuell eintragen (thematische Überschneidung mit Studentenverwaltung)
- Veranstaltungen neu anlegen
 - Dateneingabe davon abhängig machen ob es ein Seminar oder eine Vorlesung ist
 - Beschränkungen für die Klausurzulassung festlegen
 - Dozent erfragen, falls eingeloggter User kein Dozent ist
- bestehende Veranstaltung in das aktuelles Semester kopieren

Klausuren- / Übungsverwaltung(als Teil der Veranstaltungsverwaltung)

- Prüfung mit Aufgaben,Punkten und Zeitpunkt anlegen und einer Veranstaltung zuweisen
 - Entscheidung, ob es sich um eine Klausur oder ein Übungsblatt handelt
 - Dynamische Formulare abhängig von der Aufgabenzahl
- Notenschlüssel zu einer Klausur eintragen

Klausuren- / Übungsverwaltung aus Sicht des Tutors

- Gruppen anlegen und mit eigenem Kürzel versehen
 - Gruppen verschieben, zusammenführen,
 - Einzelbewertung durchführen
- Punkte pro Gruppe, Aufgabenblatt und Aufgabe erfassen (evtl. Benachrichtigung der Gruppe per E-Mail)
 - keine Punkteingabe möglich, wenn Gruppe zum Zeitpunkt des Übungsblattes leer ist (kein Student gehört zu dieser Gruppe)
- Punkte pro Student und Klausur erfassen (immer Einzelbewertung bei der Klausur)
 - vor Klausur prüfen, ob der Student die Klausurzulassung hat

Adminoberfläche

- Dozent / Tutor / Übungsleiter / Student anlegen
 - Loginvergabe

Seminar-/Praktikaverwaltung (Teil der Veranstaltungsverwaltung)

- Endnoten eintragen

Statistiken

- Notenschlüssel berechnen und eventuell exportieren
- Klausurzulassung pro Veranstaltung
- hat ein Student die Klausur oder ein bestimmtes Übungsblatt bestanden (prozentual für alle Studenten)
- pro Student und Übungsblatt / Aufgaben erreichte Punktzahlen anzeigen / für alle Studenten im Vergleich
- Notensortierte Liste der Teilnehmer
- Auswertung der Noten über die Studiengänge
- Abschneiden aller Studenten pro Übungsblatt / Aufgabe

1.4 Modularisierung

von D. Abraham, A. Brankova, S. Heider, D. Künne, K. Roehr, A. Siemer, K. Sperber

Da wir möglichst streng nach dem Model-View-Controller Design-Pattern arbeiten wollen, haben wir als Vorbereitung kleinere Arbeitspakete definiert, die jeweils von 2er- oder 3er-Teams bearbeitet werden können.

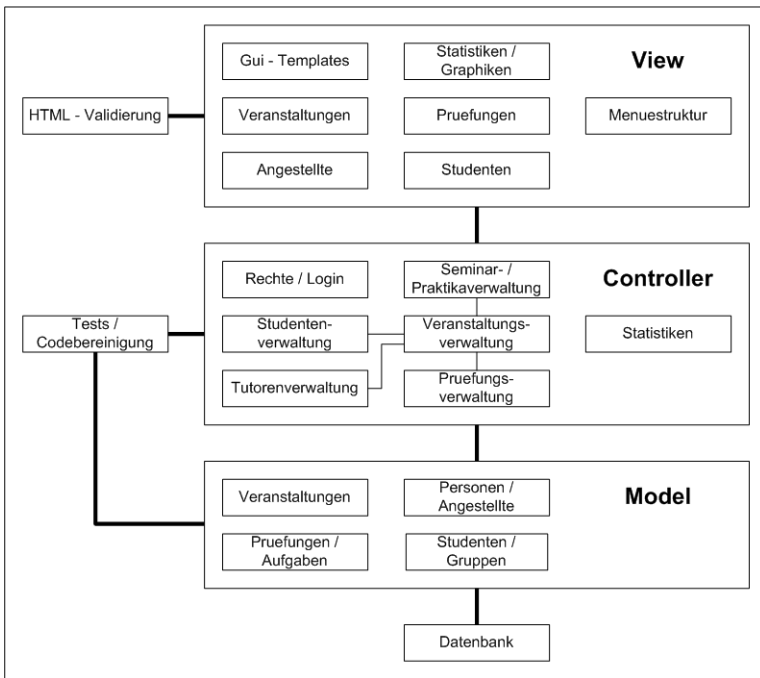


Abb. 3: MVC-Aufteilung der einzelnen Arbeitspakete

2. Planung

2.1 Layout mit CSS und jQuery, Autocomplete-Funktion und Veranstaltung-Formular

Inna Janz, Deniz Kruppe

Dynamische Effekte durch JavaScript-Libraries

Einführung

Wir stellen kurz die JavaScript-Frameworks Prototype¹, Script.aculo.us² und jQuery³ vor. Grundsätzlich ist die Funktionalität der verschiedenen Frameworks relativ ähnlich, sprich: im Prinzip kann man benutzen, was man möchte. Wir werden deshalb in jedem Framework verschiedene Effekte vorstellen, da sie mehr oder weniger analog auch in einem der anderen darstellbar wären.

Sprachunterschiede bestehen jedoch in der Syntax und vor allem in der Dateigröße. Scriptaculous-Dateien sind oftmals um einiges größer als eine entsprechende Lösung mit JQuery. Es gibt noch weitere kleine Unterschiede (zB. überlädt Prototype/Scriptaculous Standardobjekte von JavaScript, JQuery nicht), insgesamt verfolgen JQuery und Scriptaculous jedoch einen ähnlichen Ansatz. Bei der Implementierung haben wir uns für das JQuery-Framework entschieden.

Zunächst müssen die verschiedenen Libraries eingebunden werden. Während man Prototype auch allein nutzen kann, baut Scriptaculous auf Prototype auf und erweitert dieses (unter anderem um viele hübsche Grafik-Effekte). Um Scriptaculous zu nutzen, müssen also beide Libraries im Header eines HTML- bzw. PHP-Dokuments eingebunden werden (analoge Einbindung gibt es bei JQuery):

```
1 <script type="text/javascript" src="lib/prototype/prototype.js"></script>
2 <script type="text/javascript" src="lib/scriptaculous/scriptaculous.js"></script>
```

Das Document Object Model

Alle genannten Frameworks verwenden das *Document Object Model* (DOM), einen objektorientierten Ansatz der eine Schnittstelle für den Zugriff auf XML- und HTML-Dokumente spezifiziert. Die Spezifizierung besitzt verschiedene Versionen (Level), ab DOM Level 2 kann man DOM-Objekte zB. mittels der CSS-Formatierungssprache manipulieren, die später noch erwähnt wird. Zunächst werden DOM-Objekte in HTML-Containern gespeichert, die einen Identifier haben, mit dem sie nachher angesprochen werden können:

1 <http://www.prototypejs.org/>

2 <http://script.aculo.us/>

3 <http://jquery.com/>

```

1 <div id="kurzer_text">Hallo Welt!</div>
2 <div id="text_in_block" style="padding:10px; border:2px solid #ccc; background:#ffaaff;">
3   Hier steht ein Beispieltext. Ein paar Wörter,<br />
4   die von einem hübschen Kasten umrahmt werden.
5 </div>

```

Diese DOM-Objekte werden zunächst mit Hilfe des \$-Selectors ausgewählt, wobei DOM-Objekte als Knoten in Bäumen organisiert sind, und somit auch Children oder Parent-Knoten des aktuellen Objektes selektiert werden können. Danach wird die Auswahl bearbeitet.

Beispiel für Textmanipulation in Prototype

Fügt einen Beispiel-Text in ein DOM-Objekt ein.

```

1 <a href="#" onclick="$('text_in_block').insert({top:
2 '<font color=#00AA11>Dieser Text wurde vorne eingefügt.<br></font>'});">
3   Hier klicken um Text vorne einzufügen.<br />
4 </a>
5 <br />
6
7 <a href="#" onclick="$('text_in_block').insert({bottom:
8 '<font color=#FF0000><br>Dieser Text wurde hinten eingefügt.</font>'});">
9   Hier klicken um Text anzuhängen.<br />
10 </a>

```

Ein paar graphische Effekte in script.aculo.us

(für eine allgemeine Übersicht siehe hier⁴)

Erstellt eine Textbox, die flüchtet, wenn man sie anklickt:

```

1 <style type="text/css">
2   #textbox { background:#1111ff; color:#afa; padding:5px; border:1px solid #555;
3 </style>
4 <div class="bewegbare_box">
5   <a href="#" id="textbox" onclick="new Effect.Move(this, { x: 100, y: -60 });
6     return false;">
7     Klick mich!
8   </a>
9 </div>

```

Beispiele für jQuery

Aufbau der HTML Seite

```

1 <html>
2   <head>
3     <script type="text/javascript" src="lib/jquery/jquery-1.3.2.js"></script>
4     <script type="text/javascript">
5       // Hier kommt dein Code rein
6     </script>

```

⁴ <http://wiki.github.com/madrobby/scriptaculous>

```
7 </head>
8 <body>
9     // Hier kommt dein Div-Container rein
10 </body>
11 </html>
```

Für folgende jQuery UI Beispiele muss dies im head-Bereich zusätzlich eingebunden werden:

```
1 <link type="text/css" href="http://jqueryui.com/latest/themes/base/ui.all.css"
  rel="stylesheet" />
2 <script type="text/javascript" src="http://jqueryui.com/latest/ui/core.js"></script>
3 <script type="text/javascript" src="http://jqueryui.com/latest/ui/datepicker.js"></script>
4 <script type="text/javascript" src="lib/jquery/jquery.autocomplete.js"></script>
```

Beispielcode: Datepicker zeigt einen Kalender mit dem aktuellen Datum

```
1 $(document).ready(function(){
2     $("#datepicker").datepicker();
3 });
4
5 <div type="text" id="datepicker"></div>
```

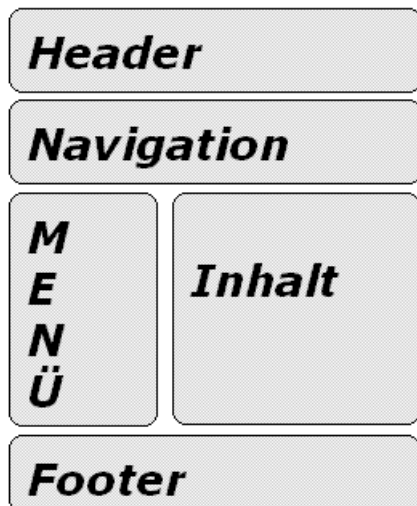
weitere Beispiele und Tutorials unter: <http://docs.jquery.com>, <http://visualjquery.com>, <http://jqueryui.com>

Layout und CSS

Layout durch das Box-Modell

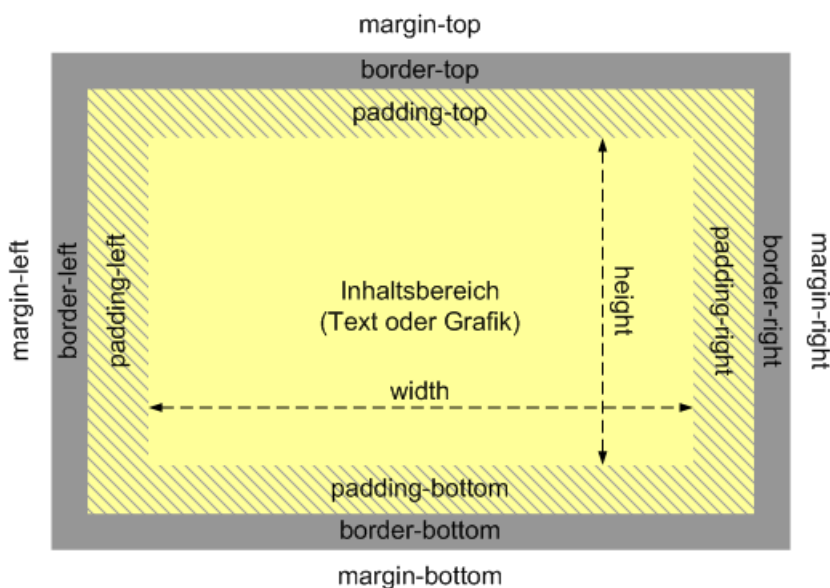
Zuerst wird das grobe Layout festgelegt, das nacher noch durch das spezielle Design verfeinert wird. Hier werden nun zunächst mal die einzelnen Strukturelemente in Div-Container abgelegt, die im vorherigen Abschnitt erwähnt wurden.

Das grobe Schema sieht nun so aus:



Die einzelnen Div-container

Es empfiehlt sich unter Umständen, eine (für den User nicht sichtbaren) Wrapper-Container zu erstellen, der dann die Einzelnen Container enthält. Die Container werden nun mit Hilfe von CSS-Stylesheets in ihren Eigenschaften (Farbe, Ausrichtung, Umrandung etc.) festgelegt. Dabei werden die Div-Container im Sinne des Box-Modells von CSS verwendet:



Das Box-Modell

CSS

Die Formatierungssprache **Cascading Style Sheets** legt eigenständige Dateien an, die im Prinzip lediglich Definitionen der eben erwähnten Boxen enthalten. Boxen werden entweder über einen eindeutigen Identifier angesprochen (in der CSS-Datei dann `#name{eigenschaften}`) oder über einen Klassennamen (es dürfen mehrere Klassenobjekte existieren, in der CSS-Datei dann `.klassenname{eigenschaften}`).

So würde beispielsweise ein "Footer"-Container im HTML-Code aussehen..

```

1 <div id="footer">
2 Hier Steht der Footer-Text, der angezeigt werden soll.
3 </div>

```

..der nun in der "main.css" definiert wird:

```

1 #footer {
2     width: 100%;
3     border-top: 1px solid;
4     padding-right: 6px;
5     background-color: #c0c0c0;
6     font-size: 100%;
7     font-style: italic;
8     font-family: Helvetica, Arial, sans-serif;
9 }

```

Außerdem wird hier die Schrift-Familie definiert (wenn mehrere angegeben werden, wird die erste, die auch installiert ist angezeigt).

Auch HTML-native Container können gestylt werden, zB. die Überschrift 2:

```

1 h2 {
2     font-size: 130%;
3     color: #665874;
4     margin: 0;
5     padding: 4px;
6     padding-bottom: 16px;
7 }

```

Das Veranstaltungs-Formular

Eine Veranstaltung kann nur von Dozenten und Übungsleitern angelegt, bearbeitet und gelöscht werden.

Beim Anlegen einer neuen Veranstaltung kann man zwischen Vorlesung und Seminar/Praktikum wählen, falls eine Vorlesung als Veranstaltungstyp gewählt wurde, wird das Formular um neue Attributfelder erweitert. Dies geschieht mit Hilfe der Methoden *show()* und *hide()* aus dem JQuery-Framework, welche das Ein- und Ausblenden von Teilstücken ermöglicht.

```

1 $('#veranstaltungstyp').change(function () {
2     if($('#veranstaltungstyp').val() == 1) {
3         $('#extra_form').show();
4     }
5     else {
6         $('#extra_form').hide();
7     }
8 }).change();

```

Smarty erlaubt eine automatische Wiedergabe des aktuellen Jahres und z.B. 4 der darauf folgenden Jahre in einer Selectbox, dieses verwenden wir für die Auswahl des Semesterjahres.

```

1 {html_select_date end_year="+4" display_days=false display_months=false}

```

Das Formular

Veranstaltung

Titel:
 Semester: SS 2009
 Max. Teilnehmer:
 Veranstaltungstyp: ---
 Übungsleiter 1:
 Übungsleiter 2:

Einfügen

Abbrechen

Veranstaltung

Titel:
 Semester: SS 2009
 Max. Teilnehmer:
 Veranstaltungstyp: Vorlesung
 Übungsleiter 1:
 Übungsleiter 2:

Vorlesung

Punkte für Zulassung: Pro Zettel
 Prozentsatz: 50 %
 Anz. Freiversuche:
 Min. Gruppengröße:
 Max. Gruppengröße:
 Anzahl Übungsblätter:
 Tutor/en: s

Hans Georg Schultz
 Benjamin Sisko

In dem Veranstaltungs-Controller wird in der Methode `new()` abgefragt, ob es sich um eine Vorlesung oder ein Seminar handelt und dann der entsprechende Typ angelegt. Außerdem können bis zu 2 Übungsleiter und mehrere Tutoren für eine Veranstaltung angelegt werden, da diese jedoch nicht in dem Datenbankschema für eine Veranstaltung aufgeführt werden, müssen diese einzeln behandelt werden.

Nachdem eine Veranstaltung erfolgreich eingetragen wurde, erscheint eine Liste mit allen Veranstaltungen, hier hat man zusätzlich die Auswahl nur die Vorlesungsliste (*vorlesung_list.tpl*) oder nur die Seminarliste (*seminar_list.tpl*) anzuschauen. Zu jedem Listeneintrag kann man sich Informationen, mit der Methode *show()* anzeigen lassen, dass Formular hierfür ist in der *vorlesung_view.tpl* bzw. *seminar_view.tpl* festgelegt, sowie einen Eintrag bearbeiten oder löschen. Beim Letzterem wird zunächst die Methode *delrequest()* im entsprechenden Model aufgerufen, welche eine Abfrage enthält, ob die Veranstaltung wirklich gelöscht werden soll. Falls dies der Fall ist, wird dann die Methode *delete()* aufgerufen.

Die Autocomplete-Funktion

Das Formular bietet die Möglichkeit Übungsleiter und Tutoren per Autocomplete anzugeben. Durch das Eingeben eines Buchstaben werden alle Vor- und Nachnamen von Studenten bzw. Angestellten aufgelistet, deren Nachname mit diesem Buchstaben beginnt.

Autocompletefunktion für Tutoren:

```
1 $('#tutor').autocomplete("autocomplete_tutor.php",
2     width: 146,
3     multiple: true,
4     matchContains: true,
5 });
```

Hier wird das Login des Tutors in ein verstecktes Feld übertragen

```
1 $('#tutor').result(function(event, data, formatted) {
2     if(data) {
3         $(document).find("input#tutor_login").val(data[1]);
4     }
5 });
```

PHP-Code dazu:

```
1 {$db = new DB($dsn);
2 $q = strtolower($_GET["q"]);
3
4 $sql = "SELECT a.Login AS Login,
5         b.Vorname AS Vorname,
6         b.Nachname AS Nachname
7         FROM Studenten AS a, Personen AS b
8         WHERE a.Login = b.Login AND b.Nachname LIKE ?";
9
10 $result = mysql_query($sql);
11 while($res=mysql_fetch_row($result)) {
12     echo $res[1]." ".$res[2]." | ".$res[0]."\n";
13 }
```

Diese Tutoren bzw Übungsleiter werden anschließend aus der Datenbank geholt und in die Vorlesungs- bzw. Seminarview eingebunden:

```
1 <tr>
2     <td><strong>Tutoren:</strong></td>
3     <td>
4         {foreach from=$tutoren item=tutor}
5             {tutor.Vorname} {tutor.Nachname}<br />
6         {/foreach}
7     </td>
```

```
8 </tr>
9
10 <tr>
11   <td><strong>Übungsleiter_1:</strong></td>
12   <td>{$leiter[0].Vorname} {$leiter[0].Nachname}</td>
13 </tr>
14 <tr>
15   <td><strong>Übungsleiter_2:</strong></td>
16   <td>{$leiter[1].Vorname} {$leiter[1].Nachname}</td>
17 </tr>
```

2.2 Template-Engines

von F.Hebestreit, P. Middendorf

Smarty

Installation

- Runterladen von <http://www.smarty.net/download.php>.
- Entpacken in Verzeichnis \$v.
- Im Projektverzeichnis Verzeichnisse *templates*, *templates_c* erstellen.
- PHP-Dateien erstellen, oben:

```
1 define('SMARTY_DIR', $v);
2 require_once(SMARTY_DIR.'/Smarty.class.php');
```

Benutzung

Ein Objekt vom Typ *Smarty* entspricht einem Smarty-Template, also pro Template ein

```
1 $smarty = new Smarty;
```

Variablenzuweisung

```
1 $smarty->assign('n', $wert);
```

wobei *n* der Name der Variable ist.

Flache Variablen

Mit `foo` greift man auf die Variable *foo* zu.

foreach

Zum Durchlaufen von assoziativen Arrays (auch numerisch wie in `array('foo','bar')`). Syntax:

```
1 <ul>
2 {foreach from=$a key=k item=v}
3 <li>{$k}: {$v}</li>
4 {/foreach}
5 </ul>
```

Wobei `$a` der Name des Arrays ist und `k` und `v` die Laufvariablen fuer den Schluessel und den Wert. Es kann noch ein `{foreachelse}` hinzugefügt werden, das ausgegeben wird, wenn das Array leer ist. Für weiteres siehe Doku⁵.

Sections

Ebenfalls zum Durchlaufen von Arrays

```
1 {section name=i loop=$mypersons}
2   <tr>
3     <td>{$mypersons[i].name}</td>
4     <td>{$mypersons[i].zipcode}</td>
5     <td>{$mypersons[i].age}</td>
6   </tr>
7 {/section}
```

Wobei `$mypersons` ein assoziatives Array ist. `section` hat Sonderbefehle wie `step` oder `start`, siehe Doku⁶.

Escaping

Will man die Smarty-Kommandos bzw. Text in geschweiften Klammern verwenden, ohne dass sie von Smarty interpretiert werden, muss man sie mit `{literal}...{/literal}` umschliessen.

HTML-Formulare

Smarty bietet Konstrukte, um aus den Variablen direkt HTML-Formularfelder zu generieren. Beispiel fuer ein Dropdownfeld:

```
1 {html_options name=foo options=$myarray}
```

Wobei `$myarray` ein Array ist, was die Optionswerte mit dem anzuzeigenden Text assoziiert. `foo` ist der Name des generierten Formularfeldes.

⁵ <http://www.smarty.net/manual/en/language.function.foreach.php>

⁶ <http://www.smarty.net/manual/en/language.function.section.php>

Anderes

Smarty verfügt noch über weitere Konstrukte, wie z.B. Bedingungskonstrukte via `{if}`, `{else}`. Man kann auch Funktionen auf die Templatevariablen anwenden, z.B. gibt `{$x | date_format "%Y"}` das Jahr des in `$x` gespeicherten UNIX-Timestamps aus.

Flexy

Installation

- Erstmal PEAR runterladen von <http://download.pear.php.net/package/PEAR-1.8.1.tgz>
- Flexy runterladen von http://download.pear.php.net/package/HTML_Template_Flexy-1.3.9.tgz
- Das Flexy-Archiv entpacken unter das Projektverzeichnis in HTML/Template
- Unter dem Projektverzeichnis Unterverzeichnisse `templates` und `templates_c` erstellen.
- PEAR irgendwohin entpacken und die PEAR.php in das Projektverzeichnis packen
- Datei `flexyutility.php` erstellen mit folgendem Inhalt:

```

1 <?php
2 require_once 'HTML/Template/Flexy.php';
3
4 class flexyutility
5 {
6     static function output($filename,$object)
7     {
8         $o = new HTML_Template_Flexy(
9             array(
10                'templateDir'   => 'templates/',
11                'compileDir'    => 'templates_c/',
12                'forceCompile' => 0,
13                'debug'         => 0,
14                'locale'        => 'en',
15                'compiler'      => 'Standard'));
16        $o->compile($filename);
17        $o->outputObject($object);
18    }
19 }
20 ?>
```

Benutzung

Erstelle Klasse `k`, die auszugebenden Daten als Instanzvariablen enthaelt. Rufe dann

```
1 flexyutility::output($dateiname,$k)
```

auf, wobei `$k` eine Instanz von `k` ist.

Flache Variablen

Auf einfache Variablen kann im Template mit `{foo}` zugegriffen werden.

foreach

Zum Durchlaufen von assoziativen Arrays. Syntax:

```
1 <ul>
2 <li flexy:foreach="mydata,key,value"> insert code here </li>
3 </ul>
```

Wobei *mydata* ein Array ist und innerhalb des Tags auf die Variablen *key* und *value* normal zugegriffen werden kann.

Anderes

- Escaping ist bei Flexy nicht noetig, weil die flexy-Kommandos sich in einem separaten HTML-Namensraum befinden.

2.3 PlotKit

von Katja Sperber

PlotKit

PlotKit ist eine Diagramm- und Graphen-Zeichnen-Bibliothek für Javascript. Die *PlotKit*-Bibliothek nutzt die *Mochikit*-Bibliothek als Basis (Version 1.3.oder höher). *MockiKit* ist wie *PlotKit* eine Open Source.(Download von *PlotKit* sowie *MochiKit*: <http://www.liquidx.net/plotkit/>)

Mit dieser Bibliothek ist es möglich einfache Balken-, Linien- und Kuchen-Diagramme darzustellen.

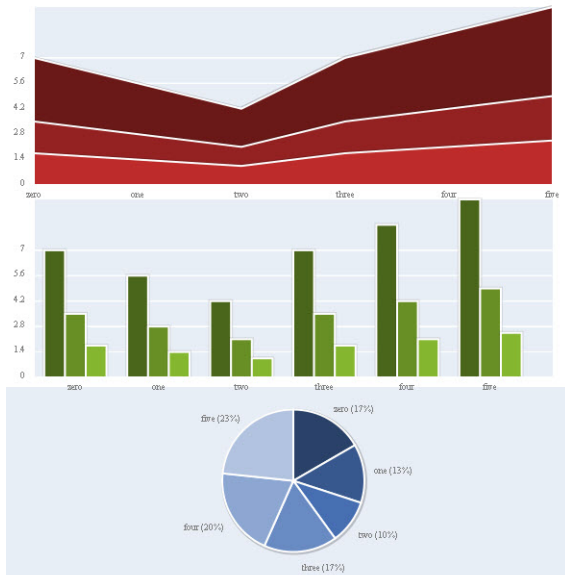


Abbildung 1

Unter den Optionen kann man sich sein Diagramm etwas anpassen. Die Standardfarbe ist normalerweise blau, doch es ist möglich andere Farben einzustellen (red: siehe Linien-Diagramm Abbildung 1 und 2; green: siehe Balken-Diagramm Abbildung 1).

```
1 "colorScheme": PlotKit.Base.palette(PlotKit.Base.baseColors()[1]) // Color : red
```

oder

```
1 "colorScheme": PlotKit.Base.palette(PlotKit.Base.baseColors()[2]) // Color : green
```

Auch der Hintergrund, die Achsen und die Achsenbeschriftung ist farblich variabel (Hintergrund: siehe Linien-Diagramm; Achsen und Achsenbeschriftung: siehe Balken-Diagramm in Abbildung 2).

```
1 "backgroundColor": Color.yellowColor(),
2 "axisLineColor": Color.redColor(),
3 "axisLabelColor": Color.blueColor()
```

Auf Wunsch können diese aber auch ganz weggelassen werden (siehe Linien-Diagramm in Abbildung 2).

```
1 "drawYAxis": false,
2 "drawYAxis": false
```

Beim Balken-Diagramm sind standartmäßig die Balken vertikal eingefügt, dies kann man aber durch eine eigene Option in horizontal ändern (PlotKit 0.9+ only; siehe in Abbildung 2).

```
1 "barOrientation": "horizontal"
```

Im Diagramm sieht das dann wie folgt aus.

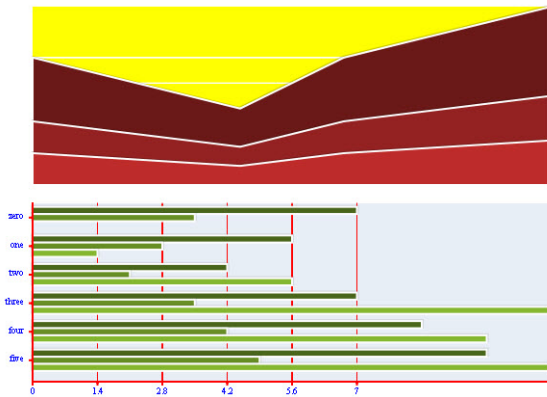


Abbildung 2

Genauso ist es möglich das Minimum und Maximum der Y- und X-Achsen einzustellen oder den Radius des Kuchendiagramms vorzugeben.

```
1 "xAxis": [1,6],
2 "yAxis": [0,3],
3 "pieRadius": 1
```

Arbeiten mit PlotKit

Um mit PlotKit zu arbeiten, muss man zu Beginn die notwendigen **Skriptdateien** einfügen. Für das Beispiel benötigen man die folgenden Dateien.

```
1 <!-- Script-File with corresponding Path -->
2 <script type="text/javascript" src="lib/mochikit/MochiKit.js"></script>
3 <script type="text/javascript" src="lib/plotkit/Base.js"></script>
4 <script type="text/javascript" src="lib/plotkit/Layout.js"></script>
5 <script type="text/javascript" src="lib/plotkit/Canvas.js"></script>
6 <script type="text/javascript" src="lib/plotkit/SweetCanvas.js"></script>
```

Desweiteren wir ein **<canvas>-Tag** benötigt, um das Diagramm an die entsprechende Stelle der Seite mit angegebener Größe einzufügen.

```
1 <div><canvas id="canvaspie" height="200" width="600"></canvas></div>
```

Zur Gestaltung des Diagramms arbeitet man dann mit den beiden Javascript-Klassen **Layout** und **Renderer**, mit denen alle weiteren Operationen durchgeführt werden können.

Die Instanzen der Klasse *Layout* kümmern sich um die Datenhaltung eines Charts.

```
1 // Layout and Data
2 // create a new layout object: first parameter = chart style, second parameter = options
3 var pie = new PlotKit.Layout("pie", options1);
4 // add a new dataset to the layout; the dataset consists of an array of (x, y) values
5 pie.addDataset("sqrt1", [[0, 5], [1, 4], [2, 3], [3, 5], [4, 6], [5, 7]]);
6 // tell the layout to calculate the layout of the chart so the renderer can draw it
7 pie.evaluate();
```

Die eigentliche Ausgabe des Charts erfolgt durch einen *Renderer*. Hierbei kann man das Format der Ausgabe wählen. Durch die Instanziierung der *Render*-Klassen entscheidet man, ob man SVG oder Canvas verwenden will. In diesem Beispiel wird mit Canvas gearbeitet.

```

1 // Renderer
2 // will get the HTML element we defined in the html-Body
3 var canvas = MochiKit.DOM.getElement("canvaspie");
4 //create the renderer: first parameter = canvas, second parameter = layout , third
  parameter = options
5 var plotter = new PlotKit.SweetCanvasRenderer(canvas, pie, options1);
6 // render the graph
7 plotter.render();

```

Zusammengeschrieben könnte das Programm folgendermaßen aussehen:

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
2 <html>
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
5     <title>Beispiel</title>
6
7     <script type="text/javascript" src="lib/mochikit/MochiKit.js"></script>
8     <script type="text/javascript" src="lib/plotkit/Base.js"></script>
9     <script type="text/javascript" src="lib/plotkit/Layout.js"></script>
10    <script type="text/javascript" src="lib/plotkit/Canvas.js"></script>
11    <script type="text/javascript" src="lib/plotkit/SweetCanvas.js"></script>
12
13    <script type="text/javascript">
14      var options1 = {
15        // Path relative to the HTML document of the iecanvas.htc file.
16        "IECanvasHTC": "/plotkit/iecanvas.htc",
17        // Color : blue
18        "colorScheme": PlotKit.Base.palette(PlotKit.Base.baseColors()[0]),
19        // X values at which ticks should be drawn
20        "xTicks": [{v:0, label:"zero"},
21                  {v:1, label:"one"},
22                  {v:2, label:"two"},
23                  {v:3, label:"three"},
24                  {v:4, label:"four"},
25                  {v:5, label:"five"}],
26        // Y values at which ticks should be drawn
27        "yTicks": [{v:0, lable:"0"},
28                  {v:1, label:"1.4"},
29                  {v:2, label:"2.8"},
30                  {v:3, label:"4.2"},
31                  {v:4, label:"5.6"},
32                  {v:5, label:"7"}],
33        "pieRadius": 0.4
34      };
35
36
37      function demo() {
38
39        var pie = new PlotKit.Layout("pie", options1);
40        pie.addDataset("sqrt1", [[0, 5], [1, 4], [2, 3], [3, 5], [4, 6], [5,
41          7]]);
42
43        pie.evaluate();
44
45        var canvas = MochiKit.DOM.getElement("canvaspie");
46        var plotter = new PlotKit.SweetCanvasRenderer(canvas, pie, options1);
47        plotter.render();
48
49      } MochiKit.DOM.addLoadEvent(demo);
50
51    </script>
52  </head>
53  <body>

```

```

54         <div id="body">
55         <h1>Style</h1>
56         <div>
57             <canvas id="canvaspie" height="200" width="600"></canvas>
58         </div>
59
60     </div>
61
62 </body>
63 </html>

```

Das Ergebnis ist dann das Kuchendiagramm aus der Abbildung 1.

Die Eingabe der (x,y)-Paare muss nicht direkt übergeben werden, sondern kann auch über ein Array oder z.B. einer Textdatei erfolgen.

Außerdem gibt es auch den sogenannten **EasyPlot**. Das ist eine Art Hülle um die eigentlichen Klassen von *PlotKit*. Hierbei werden *Layout* und *Renderer* in eine Zeile zusammengefasst. Es ist darauf zu achten, dass nun andere Skriptdateien benötigt werden.

```

1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
2  <html>
3    <head>
4      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
5      <title>EasyPlot mit Variable und txt-Datei</title>
6
7      <script type="text/javascript" src="lib/mochikit/MochiKit.js"></script>
8      <script type="text/javascript" src="lib/plotkit/excanvas.js"></script>
9      <script type="text/javascript" src="lib/plotkit/PlotKit_Packed.js"></script>
10
11     <script type="text/javascript">
12
13         var options1 = {
14             // Path relative to the HTML document of the iecanvas.htc file.
15             "IECanvasHTC": "/plotkit/iecanvas.htc",
16             // Color : blue
17             "colorScheme": PlotKit.Base.palette(PlotKit.Base.baseColors()[1]),
18             // X values at which ticks should be drawn
19             "xTicks": [{v:0, label:"zero"},
20                     {v:1, label:"one"},
21                     {v:2, label:"two"},
22                     {v:3, label:"three"},
23                     {v:4, label:"four"},
24                     {v:5, label:"five"}],
25             // Y values at which ticks should be drawn
26             "yTicks": [{v:0, label:"0"},
27                     {v:1, label:"1.4"},
28                     {v:2, label:"2.8"},
29                     {v:3, label:"4.2"},
30                     {v:4, label:"5.6"},
31                     {v:5, label:"7"}],
32             "pieRadius": 0.4
33         };
34
35         function demo() {
36
37             var data1 = [[0, 5], [1, 4], [2, 3], [3, 5], [4, 6], [5, 7]];
38             var data2 = [[0, 2.5], [1, 2], [2, 1.5], [3, 2.5], [4, 3], [5, 3.5]];
39
40             //EasyPlot: first parameter = style, second parameter = options (layout
and renderer), third
41             // parameter = the HTML element we defined in the html-Body, fourth parameter
= datasets to
42             // the layout
43             var pie = new PlotKit.EasyPlot("pie", options1, $("canvaspie"), [data1]);

```

```

44         var bar = new PlotKit.EasyPlot("bar", options1, $("canvasbar"), [data1,
data2, "sample.txt"]);
45
46     } MochiKit.DOM.addLoadEvent(demo);
47
48     </script>
49
50
51     </head>
52     <body>
53         <div id="body">
54             <h1>EasyPlot mit Variable und txt-Datei</h1>
55
56             <div id="canvaspie" style="margin: 0 auto 0 auto;" height="200"
width="600"></div>
57             <div id="canvasbar" style="margin: 0 auto 0 auto;" height="200"
width="600"></div>
58
59             </div>
60
61     </body>
62 </html>

```

Als Ergebnis kommen ähnliche Diagramme wie in Abbildung 1.

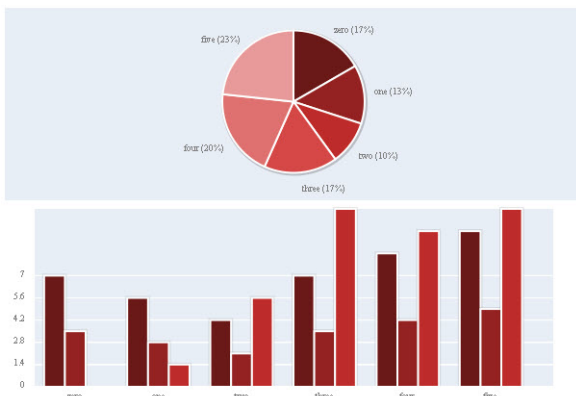


Abbildung 3

Damit *PlotKit* die (x,y)-Paarung rauslesen kann, sollte der *Sample.txt* wie folgt aussehen:

```

1 0,0
2 1,1
3 2,4
4 3,8
5 4,7
6 5,8

```

Fazit

Bei einer Meinungssuche über *PlotKit* im Internet waren die Eindrücke fast durchgehend positive. Die folgenden Vor- und Nachteile wurden dabei aufgeführt.

Vorteile:

- man kann das Diagramm ohne erneuten Seitenaufbau per Ajax rendern
- unkompliziert

- einfach interaktiv gestaltbar
- Dokumentation (<http://media.liquidx.net/js/plotkit-doc/PlotKit.html>) und QuickTutorial (<http://media.liquidx.net/js/plotkit-doc/PlotKit.QuickStart.html>) vorhanden

Nachteile:

- nur kleine, einfache Diagramme
- die Javaskript Bibliotheken können sich mit anderen Javaskripten beißen (z.B. mit jQuery bzw. jQuery Plugins)

Für uns kann dieses Programm zur visuellen Darstellung der Ergebnisse dienen wie z.B. der Vergleich der Notenverteilung von einem Jahr zu den vorherigen Jahren. Dafür müssen die auszugeben Werte nur in eine entsprechende (x,y)-Paarung gebracht werden. Einziges Problem könnten das Sich-Beissen mit anderen Javaskripten sein.

Zusatz

Bei ein dynamisches Beispiel, welches von *PlotKit* vorgegeben wird, kann man die (x,y)-Paarung direkt eingeben und die Anzahl der Paare jeder Zeit erhöhen. Genauso kann man direkt das Diagramm-Style und die Farbe wechseln.

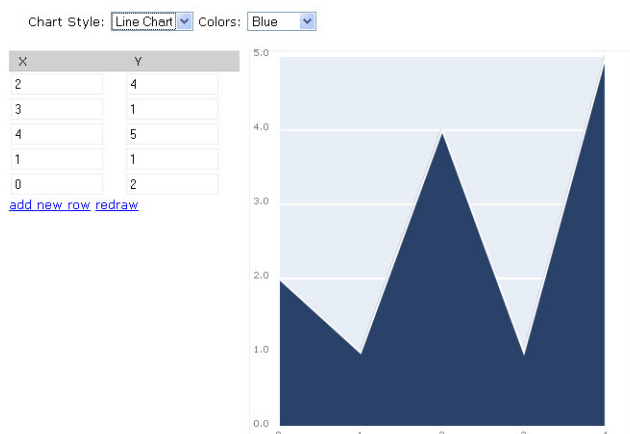


Abbildung 4

JPGraph

JPGraph ist eine objektorientierte, grapherzeugende Bibliothek für PHP. Sie ist komplett in PHP geschrieben und in allen PHP-Skripts zu benutzen (Download: <http://www.aditus.nu/jpgraph/jpdownload.php>). Es liegt eine ausführliche Dokumentation (<http://www.aditus.nu/jpgraph/docportal/classref/index.html>) und auch ein deutschsprachiges Tutorial (<http://www.binnendijk.net/jpgraph/index.php?page=startseite>) vor. In dem Tutorial werden ausführlich alle Schritte von der Installation bis hin zu einem ersten Einstieg genau erläutert.

Vorteile:

- größeres Diagrammangebot, d.h. nicht nur die einfachen Linien-, Balken- und Kuchen-Diagramme, sondern z.B. auch 3D Pies, mehrere Pies in einer Grafik, Gantt-Diagrammen oder Odometers (Tachometer-ähnliche Grafiken)
- man kann die Bilder mit der *JPGraph*-Funktion *Stroke()* temporär speichern und damit cachen

Nachteile:

- das Aussehen der Kuchendiagramme und das der Tachometer da die Rundungen Ecken haben
- der riesige Codeoverhead der Bibliothek.
- so richtig schön sehen die Diagramme erst nach grossen Aufwand aus
- Trennung nach php 4 und 5

Basis-Diagramme:

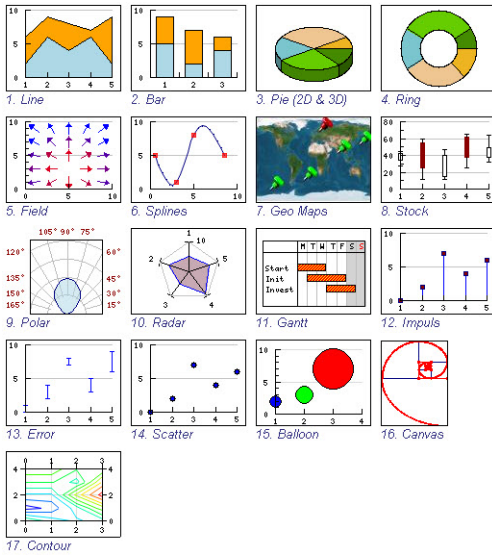


Abbildung 5



18. Antispam

Abbildung 6

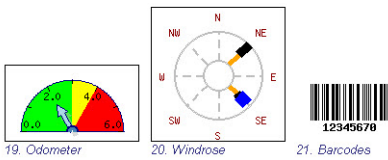


Abbildung 7

3. Login, Menü & Statistiken

3.1 Grundgerüst und Login

von F.Hebestreit, P. Middendorf, D.Künne

Konzept und Dateisystem

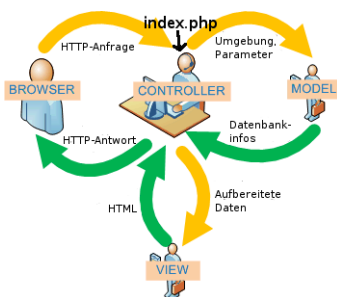


Abb. 4: MVC

Das System orientiert sich am Model-View-Controller-Konzept. Die Modelklassen entsprechen den Entitäten bzw. Beziehungen der Datenbank, die Controller benutzen die Modelklassen und geben die aufbereiteten Daten an die Templates weiter. Diese werden geparkt und geben einen String zurück, der an den Client geschickt wird.

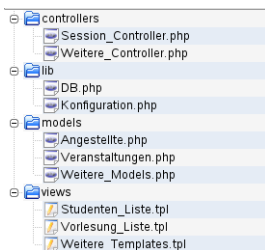


Abb. 5: Das Tuks-Dateisystem

Im Grundordner befindet sich der Einsprungpunkt *index.php*. Die Models befinden sich in *models/*, die Controller in *controllers/* und die Bibliotheken und global verfügbaren Dateien in *lib/*. Die Templatedateien befinden sich in *views/*.

Mechanik

Eine URL in Tuks hat immer die Form:

```
1 index.php?controller=a&action=b
```

Hierbei bezieht sich *a* auf eine Klasse aus dem Verzeichnis *controllers/*. Wird kein Parameter angegeben, wird der *Session_Controller* geladen, der sich um das Ein- und Ausloggen kümmert. Der Parameter *action* gibt an, welche Methode aus dem Controller aufgerufen wird. Wird *action* nicht angegeben, wird eine Standard-Action aufgerufen. Es können natürlich noch weitere Parameter in der URL übergeben werden.

Controller-Basisklasse, Rechte

Jeder der Controller erbt von der abstrakten Klasse *Controller*, die vereinfacht wie folgt aussieht:

```
1 abstract class Controller
2 {
3     protected $env;
4     protected $rechte;
5
6     function hole_resultat();
7     abstract function action_();
8 }
```

Die Klasse enthält ein Umgebungsobjekt, was die Datenbankverbindung enthält sowie den momentan eingeloggtten User. So können alle abgeleiteten Controller komfortabel darauf zugreifen.

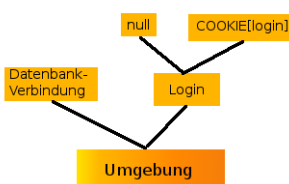


Abb. 6: Die Umgebungsstruktur

Die Variable *\$rechte* ist ein Array, was einer Controller-Aktion die Rechte zuordnet, die der User haben muss, um die Aktion auszuführen. Die Klassen, die von Controller erben, füllen das Array mit sinnvollen Werten, beispielsweise:

```
1 $rechte =
2     array(
3         'zulassungen_anzeigen' => array(RECHTE_DOZENT, RECHTE_UEBUNGSLEITER),
4         'leistung_anzeigen' => array(RECHTE_DOZENT, RECHTE_UEBUNGSLEITER, RECHTE_TUTOR));
```

Hier kann die Aktion "zulassung_anzeigen" also nur ausführen, wenn man Dozent oder Übungsleiter ist. Die Aktion "leistung_anzeigen" kann man auch als Tutor ausführen.

Die Methode *hole_resultat* der Controller-Basisklasse sieht vereinfacht so aus:

```
1 function hole_resultat()
2 {
3     if (!isset($action))
```

```
4     $action = '';
5
6     $s = schnittmenge($rechte[$action],meine_rechte($vorlesung));
7
8     if (leere_menge($s))
9         throw new Exception('Unzureichende Rechte');
10
11     ('action_'. $action)();
12 }
```

Die Methode *meine_rechte* bekommt die aktuell ausgewählte Vorlesung übergeben und bezieht diese in das Ergebnis mit ein, sodass beispielsweise ein Tutor nur in seinen eigenen Vorlesungen Aktionen durchführen kann. Sind keine Rechte für eine Aktion definiert und soll diese Aktion ausgeführt werden, gibt es eine Fehlermeldung als Sicherheitsmaßnahme.

Damit obige Methode auch im Fall einer leeren Variable *\$action* funktioniert gibt es die abstrakte Methode *action_*. Sie repräsentiert die besagte Standard-Action. Jeder abgeleitete Controller muss selber entscheiden, was in dem Fall zu tun ist, aber meistens ist das Weglassen einer Action einfach ungültig.

Die *action_*-Methoden, die von *hole_resultat* aufgerufen werden, geben entweder einen String zurück, der dann in den Inhaltsbereich der Seite eingefügt wird, oder ein Weiterleitungsobjekt, das eine Weiterleitungs-URL enthält. Auf diese Art und Weise können Controller aufeinander oder auf andere Aktionen verweisen; ein Controller zum Anlegen eines Studenten könnte nach dem Anlegen eine Liste von allen Studenten anzeigen beispielsweise.

Model

Das Interface für Model sieht wie folgt aus:

```
1 abstract class Model
2 {
3     protected $env;
4
5     abstract function insert();
6     abstract function delete();
7 }
```

Auch das Model benötigt die Datenbankverbindung sowie den eingeloggten User und bekommt somit vom Controller das Umgebungsobjekt.

Die abgeleiteten Models (beispielsweise *Person*) bekommen alle ihre Daten im Konstruktor übergeben. Daraufhin kann man *insert* aufrufen um das Model zu "instanziiieren". Auf ein bereits instanziiertes Model kann man *delete* aufrufen um diese wieder zu löschen. Updates werden von den abgeleiteten Models selber geregelt. Beispiel:

```
1 $p = new Person("pmiddend", "Phlpp", "Middendorf", ...);
2 $p->insert();
3 $p->Vorname = "Philipp";
4 $p->update();
```

Fazit

Das vorgestellte System hat einige Vorteile gegenüber eines naiveren Ansatzes. Um neue Funktionalität einzubinden - also etwa einen neuen Controller - reicht es aus, eine neue Klasse hinzuzufügen und diese von *Controller* ableiten zu lassen. Auch die Rechteverwaltung wird dem Programmierer komplett abgenommen und in eine tiefere Programmebene verschoben.

Login

Um eine Sitzung in Tuks aufrecht zu erhalten werden Cookies genutzt. Man meldet sich in einem Formular mit Name und Passwort an, woraufhin der *Session_Controller* mit der Aktion *action_login* aktiviert wird. Dieser Controller sieht vereinfacht so aus:

```
1 class Session_Controller extends Controller
2 {
3     function action_login()
4     {
5         if (anzahl_reihen(sql("SELECT COUNT(*) FROM Personen WHERE Login=? AND Passwort=MD5(?)))
6         > 0)
7         {
8             setze_cookie('login',array($name,md5($passwort)),aktuelle_zeit() + 2*3600);
9             return startseite();
10        }
11        else
12            error("Falscher Benutzername bzw. falsches Passwort!");
13    }
14    function action_logout()
15    {
16        lösche_cookie('login');
17        return startseite();
18    }
19 }
```

Dieser überprüft, ob es den Nutzer mit dem Passwort in der Datenbank gibt und füllt daraufhin den Cookie namens *login* mit einem Array bestehend aus dem Login und dem verschlüsseltem Passwort. In *index.php* wird dann bei jedem Seitenaufruf ein Abgleich mit der Datenbank und dem Cookie durchgeführt. Dies muss geschehen, weil die Cookies clientseitig gespeichert werden, also die Möglichkeit von Missbrauch besteht.

3.2 Generierung des Menüs

von F.Hebestreit, P. Middendorf, D.Künne

Motivation

Ein Benutzer kann sowohl mehr als eine Rolle haben als auch mehr als eine Veranstaltung besuchen. Jemand könnte z.B. ein Seminar besuchen und gleichzeitig Tutor in einer Vorlesung sein. In seiner Rolle als Tutor hat er aber andere Aktionen zur Verfügung als im Seminar. Man muss also in Rollen und Vorlesungen komfortabel und schnell navigieren können und es soll jeweils eine Liste von den Dingen angezeigt werden, die für die aktuelle Auswahl relevant ist.

Man müsste also zwei Formulare abschicken, im ersten Schritt wählt man seine Rolle. Daraufhin errechnet der Server, in welchen Vorlesungen der User in der Rolle eingetragen ist. Im zweiten Schritt wird die Vorlesung ausgewählt und der Server berechnet, welche Aktionen der User ausführen kann.

Umsetzung

The screenshot shows a web interface with two main sections. The top section, titled 'Rolle und Veranstaltung', contains two dropdown menus. The first dropdown, labeled 'Rolle', has 'Dozent' selected. The second dropdown, labeled 'Veranstaltung', has 'Backen ohne Fett' selected. The bottom section, titled 'Meine Aktionen', lists various actions grouped by category: 'Veranstaltungen' (Veranstaltung anlegen, Veranstaltungen anzeigen, Veranstaltung bearbeiten, Angestellter anlegen), 'Prüfungen' (Klausur erstellen, Prüfungen anzeigen, Übungsblatt erstellen, Punkte für Übungsblätter einfügen, Gruppen verwalten), 'Personen' (Studenten verwalten), 'Noten' (Endnoten anzeigen, Endnote eintragen), and 'Statistik' (Übungsblattstatistik, Klausurstatistik, Zulassungen anzeigen, Auswertungen anzeigen).

Abb. 7: Vorlesung ist ausgewählt

The screenshot shows a web interface similar to the previous one. The 'Rolle und Veranstaltung' section has 'Dozent' selected for 'Rolle' and 'Seminar' selected for 'Veranstaltung'. The 'Meine Aktionen' section is now filtered to show only the 'Noten' category, which includes 'Endnoten anzeigen' and 'Endnote eintragen'.

Abb. 8: Seminar ist ausgewählt

Um Zeit und Traffic zu sparen, werden die oben beschriebenen zwei Formulare zu einem einzigen, dynamischen Formular zusammengefasst. Hierzu wird AJAX eingesetzt. Nach dem Einloggen auf der Seite wird dem Benutzer eine Liste seiner Rollen mitgegeben. Diese werden in ein Auswahlfeld eingefügt. Wählt der Benutzer eine bestimmte Rolle aus, wird mit Hilfe von Javascript eine Anfrage an den Server gesendet, die dazugehörigen Vorlesungen zu schicken. Als Antwort schickt der Server eine XML-Datei, die beispielsweise so aussieht

```

1 <vorlesungen>
2   <vorlesung>
3     <verid>1</verid>
4     <typ>Seminar</typ>
5     <titel>Backen ohne Fett</titel>
6   </vorlesung>
7   <vorlesung>
8     <verid>2</verid>
9     <typ>Vorlesung</typ>
10    <titel>Die Zahl 42</titel>
11  </vorlesung>
12 </vorlesungen>

```

Der Code zur Verarbeitung dieser XML-Datei sieht vereinfacht so aus:

```
1 // In "meinSelect" ist ab sofort das Auswahlfeld für die Vorlesungen enthalten.
2 var meinSelect = document.getElementById(formular).verid.options;
3 // Lösche alle bisherigen Feldeinträge
4 loescheAlleSelect(document.getElementById(formular).verid);
5
6 // Hole alle Tags mit Namen "vorlesung". getElementsByTagName liefert diese als Array
  zurück.
7 var vorlesungen = doc.getElementsByTagName("vorlesung");
8
9 // In dem Auswahlfeld werden nur Titel und die VerID gespeichert,
10 // wir müssen uns den Typ der Veranstaltung separat merken (den
11 // brauchen wir, um die verfügbaren Aktionen dynamisch zu generieren)
12 veranstaltungsListe = new Array();
13
14 var gueltig = false;
15 for (var i = 0; i < vorlesungen.length; ++i)
16 {
17     veranstaltungsListe.push(
18         vorlesungen[i].getElementsByTagName("typ")[0].childNodes[0].nodeValue);
19
20     // Hier wird die Vorlesung dem Dropdown hinzugefügt. Angezeigt wird der Titel, dahinter
21     // steckt die VerID
22     meinSelect[meinSelect.length] =
23         new Option(
24             vorlesungen[i].getElementsByTagName("titel")[0].childNodes[0].nodeValue,
25             vorlesungen[i].getElementsByTagName("verid")[0].childNodes[0].nodeValue);
26
27     // in "aktuelleVerID" steckt die vorher ausgewählte VerID, siehe unten für eine Erklärung
28     if (vorlesungen[i].getElementsByTagName("verid")[0].childNodes[0].nodeValue ==
29         aktuelleVerID)
30     {
31         meinSelect.selectedIndex = meinSelect.length-1;
32         gueltig = true;
33     }
34 }
35 if (!gueltig)
36 {
37     meinSelect.selectedIndex = 0;
38     verid = (meinSelect.length > 0 ? meinSelect[0].value : -1)
39 }
40
41 // Hier wird anhand der aktuellen Auswahl das Menü mit den Aktionen generiert.
42 generiereMenue();
```

Wird eine bestimmte Veranstaltung ausgewählt, wird wieder *generiereMenue()* aufgerufen, da sich die Aktionen evtl. geändert haben (beim Wechsel von Seminar zu Vorlesung beispielsweise).

Lädt man die Seite neu oder klickt man auf einen Aktionslink, soll im Menü die letzte Rolle-Vorlesung-Kombination erhalten bleiben. Dies kann man entweder dadurch erreichen, dass man diese Daten immer in der URL weitergibt, oder man speichert sie einfach in einem Cookie. Cookies werden ohnehin benötigt, da man sich ohne sie nicht einloggen kann.

Laden des Menüs

Beim Laden des Menüs wird also die Variable *aktuelleVerID*, sowie *aktuelleRolle* mit den Werten aus dem Cookie gefüllt, falls schon einer vorhanden ist. Ansonsten wird *aktuelleRolle* mit der ersten Rolle gefüllt, die in der Liste eingetragen ist. Anhand *aktuelleRolle* wird dann die Vorlesungsliste beschafft und (siehe Code oben) überprüft, ob *aktuelleVerID* einen gültigen Wert besitzt (ein Benutzer könnte aus einer Veranstaltung ausgetragen sein, obwohl sein Cookie noch die VerID davon gespeichert hat). Ansonsten wird hier analog die erste Veranstaltung aus der Liste ausgewählt. Diese beiden Werte werden dann im Cookie "verewigt".

Aktionsmenü

Das Aktionsmenü wird nicht vom Server generiert, sondern ist statisch im Client (genauer, im Template `menue.tpl`) kodiert. Es spricht allerdings nichts dagegen, als Verbesserung wieder AJAX zu nutzen und sich das Menü vom Server als XML-Datei zu beschaffen. Als Datenstruktur für die Aktionen wird ein mehrdimensionales, assoziatives Array verwendet. Es wird hier ein Rollenstring verbunden mit einem Array von Aktionen. Eine Aktion besteht wiederum aus einem Veranstaltungstyp (Seminar, Vorlesung) und einem Link. Im Typ kann auch noch das Flag "Überschrift" übergeben werden um anzudeuten, dass dieser Menüeintrag später nicht eingerückt werden soll, sondern eben als Überschrift dient. Das Studentenmenü hat z.B. folgende Form:

```
1 // Das wird an die URL angehängt, damit die Controller damit arbeiten können.
2 var optionen = "&verid="+verid+"&rolle="+rolle;
3
4 var ar = new Array();
5 ar["Tutor"] = new Array();
6 ar["Dozent"] = new Array();
7 ar["Student"] = new Array();
8 ar["Uebungsleiter"] = new Array();
9
10 // ...
11
12 ar["Student"].push(
13     new Array(
14         "Vorlesung,Ueberschrift",
15         "<strong>Veranstaltungen</strong>"));
16 ar["Student"].push(
17     new Array(
18         "Vorlesung",
19         "<a href=\"index.php?controller=veranstaltung&action=show&"+optionen+"\">Veranstaltung
20         ansehen</a>"));
21 // hier weitere Einträge
```

Und durchlaufen wird das Array wie folgt:

```
1 for (var i = ar[v].length-1; i >= 0; --i)
2 {
3     // In x befindet sich danach der Link für die aktuelle Aktion und die aktuelle Rolle
4     var x = ar[v][i][1];
5     // Den Veranstaltungstyp splitten wir am Komma
6     var splitted = ar[v][i][0].split(',');
7
8     // Hat man hier nur ein Element, also nur einen Veranstaltungstyp, keine Überschrift,
9     dann
10    // muss hier eine Einrückung stattfinden
11    if (splitted.length == 1)
12        x = "<ul style=\"list-style:none\"><li>"+x+"</li></ul>";
13
14    // Füge dies dem Menü hinzu, wenn der Veranstaltungstyp stimmt.
15    if (splitted[0] == veranstaltungstyp)
16        fuegeLiAn(
17            container,
18            x);
19 }
```

3.3 Statistiken

von F.Hebestreit, P.Middendorf, D.Künne

Vorgaben

Die Statistiken innerhalb von Tuks sollen mindestens genau die gleiche Aussagekraft haben, wie die bisher verwendeten. Diese wurden bisher mittels Skripten aus den Punktedateien der Tutoren generiert und grafisch aufbereitet.

Dabei ist zwischen zwei Auswertungen zu unterscheiden: Zum Einen die Auswertung, die nach Abschluss der Testate geführt wird und anzeigt, welche Studenten eine Zulassung zur Klausur haben und wie sie im Testbetrieb abgeschnitten haben. Die zweite Auswertung erfolgt jeweils nach einer Klausur. Hier werden ähnliche Daten wie bei den Testaten ausgewertet.

Als erstes soll angezeigt werden, mit welchem Erfolg die einzelnen Aufgaben bearbeitet wurden. Diese Auswertung erfolgt einmal als Tabelle und einmal anhand einer Grafik.

Aufgabe	1	2	3	4	5	6	7	8	9	10
Abgabe	77	77	74	76	75	77	77	74	68	76
Punkte	55%	42%	40%	34%	67%	77%	83%	28%	20%	79%

Abb. 9: Abgabe pro Aufgabe

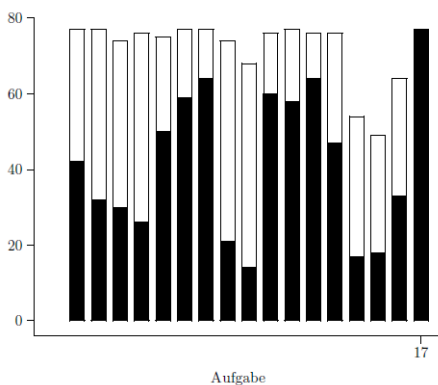


Abb. 10: Abgabe + Prozent

Des Weiteren soll angezeigt werden, wie viele Studenten mit der gleichen Punktzahl eine Klausur / die Übungsblätter bearbeitet haben. Hierbei wird die Auswertung einmal für alle Studenten erstellt und anschließend noch nach Studiengängen getrennt.

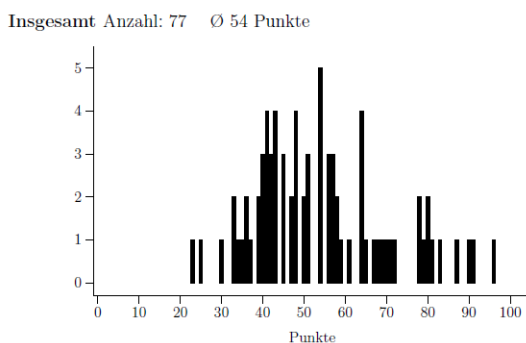


Abb. 11: Verteilung der Punkte über alle Teilnehmer

Bei den Klausuren muss zusätzlich noch eine Auswertung erfolgen, aus der der Aushang generiert werden kann. Hier muss zur Matrikelnummer eines Studenten seine Note und sein ECTS-Grade vermerkt werden. Außerdem wird noch eine Bestenliste erzeugt, die die Studenten nach absteigender Punktezahl sortiert ausgibt.

Umsetzung

Anzeige abhängig von Login

Da es möglich sein sollte, dass sich verschiedene Benutzer des Systems die Statistiken anzeigen lassen können, muss eine Filterung auf die anzuzeigenden Daten möglich sein.

- Benutzer mit Rolle "Student"
 - darf sich seine Endnote in einer Klausur / Prüfung / Seminar anschauen
 - darf eine Auswertung seiner Übungsblätter anschauen
- Benutzer mit Rolle "Tutor"
 - darf sich die Endnoten seiner Tutanden in einer Klausur / Prüfung / Seminar anschauen
 - darf eine Auswertung seiner Tutanden über die Übungsblätter anschauen
 - darf sich die Zulassungen seiner Tutanden in der betreuten Veranstaltung anzeigen lassen
- Benutzer mit Rolle "Übungsleiter" / "Dozent"
 - alle Ansichten eines Tutors für alle eingetragenen Teilnehmer in der Veranstaltung

Um diese Filterung durchzuführen und möglichst nicht für jede Rolle eine eigene Methode / eine eigene SQL-Query schreiben zu müssen, verwenden wir die IN-Klausel von SQL. Der Methode wird hierfür ein Array mit allen Logins übergeben, für die sich der angemeldete Benutzer Statistiken anzeigen lassen darf. Im SQL-Statement wird dann abgefragt, dass in der Ergebnismenge nur Einträge zu den übergebenen Logins enthalten sind.

```
1 SELECT
2   Vorname, Nachname, Note
3 FROM
4   Personen, pruefen
5 WHERE
6   Personen.Login = pruefen.StudLogin AND
7   VerID = ? AND
8   StudLogin IN (join($logins, "',''))
```

Grafische Aufbereitung der Daten

Ein Großteil der Daten wird lediglich in Tabellenform ausgegeben und mit Links zu weiterführenden Informationen versehen. Wie in der beispielhaften Abbildung 12 zu sehen ist, wird für alle in die Vorlesung eingetragenen Studenten angezeigt mit welchem Erfolg sie an den einzelnen Übungsblättern oder Klausuren teilgenommen haben. Über den angezeigten Prozentsatz gelangt man zu einer weiteren Ansicht, in der für den gewählten Studenten die Punkte aufgabenweise aufgelistet werden. Die explizite Generierung einer Bestenliste entfällt somit, da es in diesen Tabellen problemlos möglich ist nach den Werten in den einzelnen Spalten zu sortieren.

Auswertung der Übungsblätter in Veranstaltung "Algorithmen"

Name	Blatt 1	Blatt 2
Of Nine, Seven	100%	
Burkhalter, Albert		
Sisko, Benjamin	0%	
Carter, Andrew		
Tucker, Charles	100%	
Archer, Jonathan	100%	
Kirk, James Tiberius		
Picard, Jean-Luc	100%	
Tucker, Charles	100%	
Schultz, Hans Georg		

Abb. 12: Anzeige der Übungsblätter

Des Weiteren gibt es eine Liste in der angezeigt wird, welcher Student zum aktuellen Zeitpunkt über eine Klausurzulassung verfügt (Abbildung 13) und eine Liste, die die Endnote und den ECTS-Grade eines Studenten anzeigt (Abbildung 14).

Klausurzulassungen in der Veranstaltung "Algorithmen"

Student	MatrNr	Zulassung
Archer, Jonathan	136325	Ja
Maus, Micky	121115	Nein
Of Nine, Seven	139825	Ja
of Reading, Maven	121112	Nein
Paris, Tom	111114	Nein
Picard, Jean-Luc	121118	Ja

Abb. 13: Liste mit Zulassungsinformationen

Notenliste im Seminar/Praktikum "Seminarrest"

Student	eingetragen von	Note	ECTS-Grade
Burkhalter, Albert	Varchar, Willi	1.7	B
Carter, Andrew	Varchar, Willi	2.0	B
Cochrane, Zephram	Varchar, Willi	2.3	C
Hill, Terence	Varchar, Willi	5.0	F

Abb. 14: Notenliste eines Seminars

Da die erzeugten Grafiken möglichst nah an den Vorgaben sein sollten, haben wir uns gegen PlotKit und für JGraph entschieden. Außerdem wird durch den Einsatz von PHP zur Erzeugung der Grafiken auch Rechenlast von den Clients genommen, da hier die Bilder bereits komplett auf dem Server erzeugt werden. Da sich einige der Grafiken sehr ähnlich sind, haben wir pro Grafiktyp eine Vorlage verwendet, die dann dynamisch mit den anzuzeigenden Werten befüllt wird. Dafür wird diese PHP-Datei als src-Attribut eines image-Tags eingebunden und die benötigten Daten werden an diesen Aufruf angehängt.

```
1 
```

Aus diesen übergebenen Daten wird dann mit der JGraph-Library das gewünschte Diagramm erzeugt (vergleiche Abbildung 10 und Abbildung 11)

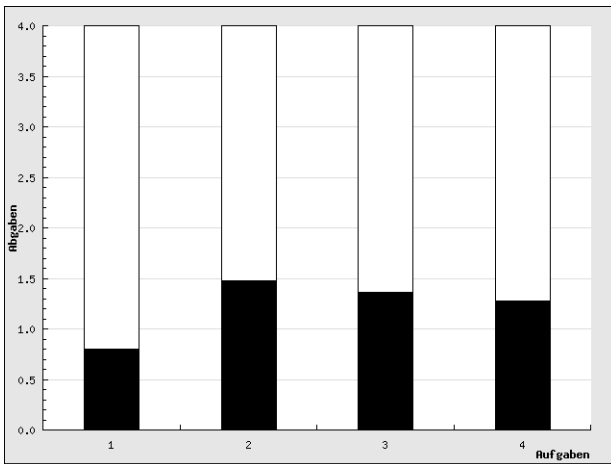


Abb. 15: akkumuliertes Diagramm aus zwei Datenreihen

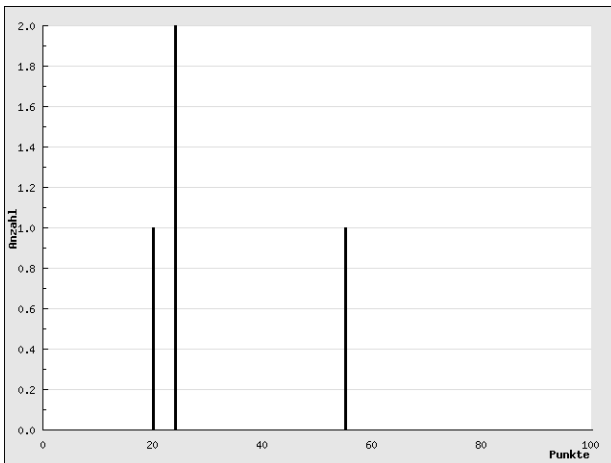


Abb. 16: einfaches Balkendiagramm

Probleme

Bei der Umsetzung zeigte sich, dass das ursprüngliche Datenbankdesign nicht ganz ausreichte um alle Daten sinnvoll aufbereiten zu können. Aus diesem Grund wurden die *Studiengaenge* als eigene Entität aus den *Studenten* ausgelagert und die Entitäten *Notenschlüssel* und *ECTS-Grade* erzeugt. Letzteres wurde notwendig, da wir uns dazu entschieden haben, die Berechnung der Noten und des ECTS-Grades in der Datenbank zu machen. Dies stellt eine enorme Performanceverbesserung im Vergleich zu einer Berechnung mittels PHP dar.

4. Studenten- und Gruppenverwaltung

- Studentenverwaltung
 - Frontend / Bedienung
 - Features
 - Einzelnen Student hinzufügen / Student editieren
 - Detailseite eines Studenten
 - Matrikelhistorie
 - Mehrere Studenten hinzufügen / csv-Format
 - Backend / Code
 - Student.php
 - Student_Controller.php
 - CSVImport.php
- Gruppenverwaltung
 - Konzept
 - Frontend / Bedienung
 - Features
 - Übersicht aller Gruppen einer Vorlesung
 - Gruppe zu einer Vorlesung anlegen
 - Gruppe bearbeiten
 - Detailansicht einer Gruppe
 - Student zu einer Gruppe hinzufügen
 - Details zu einem Studenten ändern
 - Backend / Code
 - Die Modelklasse Gruppe.php
 - Die Controller Klasse Gruppen_Controller.php

Studentenverwaltung

Nico Marniok

Frontend / Bedienung

Features

- Student aus csv-Datei importieren.
- Einzelne Studenten hinzufügen.
- Mehrere/Einzelne Studenten einer Veranstaltung hinzufügen.
- Matrikelhistorien verwalten.
- Daten einzelner Studenten ändern.
- Studentendetails anzeigen.
- Nach Studenten suchen.
- Studenten können ihr Passwort und ihre E-Mailadresse ändern.
- Voller Zugriff nur als Dozent



Abbildung 1.1

Einzelnen Studenten hinzufügen / Student editieren

- **Login kann nur einmal beim Einfügen gesetzt werden (da Schlüssel)**
- Mehrere Studiengänge können mit Semikolon getrennt werden
- Das Passwort wird nur hier in Klartext angezeigt, später kann es nicht rekonstruiert werden
- Es besteht für den Studenten die Möglichkeit sein eigenes Passwort zu ändern
- Das Passwort wird nicht angezeigt. Wenn das Feld leer gelassen wird, bleibt es gleich.

Abbildung 1.2

Abbildung 1.3

Detailseite eines Studenten

- Wird durch einen Klick auf das blaue *i* in der Studentenliste erreicht
- Alle Details eines Studenten werden angezeigt
- Die betreuten Vorlesungen, besuchten Veranstaltungen und seine Gruppen sind auf die jeweiligen Detailseiten verlinkt
- Neben der Matrikelnummer des Studenten ist der Link 'Historie', der zur Matrikelhistorie des Studenten führt

Matrikelhistorie

- Wird erreicht durch die Detailseite eines Studenten
- Hier können die Matrikelnummern eines Studenten betrachtet und manipuliert werden

Mehrere Studenten hinzufügen / CSV-Format

- csv Dateien aus studIP werden unterstützt:

- Komplettes Format: Titel Vorname Nachname Titel2 Privatadr Privatnr E-Mail Anmeldedatum Kontingent Studiengänge Matrikelnummer Studieninfo synced Geschlecht „Mitglied der Gruppen“ Sprache Bemerkung
- Verwertete Informationen: Titel, Vorname, Nachname, E-Mail, Matrikelnummer, Studieninfo
- Generierte Informationen:
 - Login: E-Mail Adresse wird auf @uos.de oder @uni-osnabrueck.de geprüft und der Präfix zum Login gemacht. Sollte keine solche E-Mail Adresse vorhanden sein wird der Login studIP-typisch aus dem ersten Buchstaben des Vornamens und den 7 ersten Buchstaben des Nachnamens generiert
 - Passwort: Wird die Matrikelnummer
- Informationen über die Erfolgsrate des Importierens werden danach angezeigt (Bereits vorhandene Logins, etc...)
- Während des Importierens können die generierten Studenten bereits in eine Veranstaltung eingetragen werden
- Während des Importierens können die Studenten einer Veranstaltung hinzugefügt werden

Beispiel Attach:

- Situation:
 - Die csv-Datei enthält 8 Studenten, von denen 4 bereits in der Datenbank stehen
 - 2 der 4 besuchen bereits eine Vorlesung, in die alle Studenten der csv-Datei eingetragen werden sollen

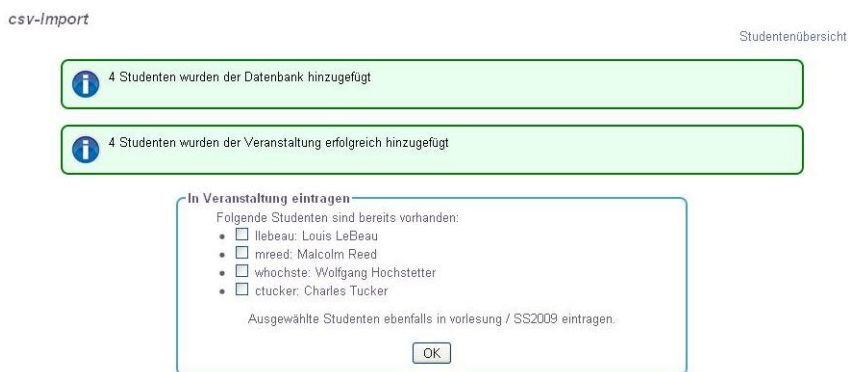


Abbildung 1.4

- Unten stehen die Studenten, die bereits in der Datenbank sind
- Hier kann man nun diejenigen auswählen, die ebenfalls in die Veranstaltung eingetragen werden sollen
- Am Ende erhält man eine Meldung darüber, welche Studenten nicht in die Veranstaltung eingetragen werden konnten

Backend / Code

Student.php

Die Klasse Student enthält alle Low Level Operationen auf den Studenten, wie Einfügen, Ändern, alle Studenten beschaffen, etc...

```

1 class Student extends Model {
2     /* Instanzvariablen... */
3     /* Konstruktor, Update, Insert, Delete, Getter, Setter*/
4
5     /**
```



```

6     * Prüft ob das übergebene Passwort mit dem des Studenten übereinstimmt und liefert
    true oder false
7     */
8     public function check_passwort($pass) { /*...*/ }
9
10    /**
11     * Statische Klassenfunktion, die einen Studenten anhand seines logins als Objekt
    zurückliefert.
12     */
13    public static function get_student($login, $env) { /*...*/ }
14
15    /**
16     * Liefert die Studenten, deren logins im übergebenen array stehen
17     */
18    public static function get_studenten($logins, $env) { /*...*/ }
19
20    /**
21     * Liefert alle Studenten
22     */
23    public static function get_alle_studenten($env) { /*...*/ }
24 }

```

Student_Controller.php

Die Klasse Student_Controller enthält alle Funktionen, die Daten beschaffen und an die View übergeben. Dafür werden diverse actions zur Verfügung gestellt. Beispiele:

```

1  /**
2   * Zeigt die Studentenliste an
3   */
4  public function action_list() { /*...*/ }
5
6  /**
7   * Verarbeitet die hochgeladene CSV-Datei und zeigt anschließend eine Zusammenfassung
8   */
9  public function action_csv() { /*...*/ }
10
11  /**
12   * Zeigt einen Bestätigungsdialog, um mehrere Studenten zu löschen
13   */
14  public function action_deletemultiplerequest() { /*...*/ }
15
16  /**
17   * Löscht mehrere Studenten und zeigt danach die Studentenliste
18   */
19  public function action_deletemultiple() { /*...*/ }

```

CSVImport.php

Da die Funktionen des CSV-Importes gänzlich anders sind als der Rest des Programms, sei er hier aufgeführt. Die Klasse bekommt bei der Konstruktion das Environment Objekt übergeben, sowie den Pfad der CSV-Datei, die es zu importieren gilt. Die Funktion *import()* startet schließlich den Vorgang. Es wird Zeile für Zeile ausgelesen und der Inhalt entsprechend einiger Heuristiken verarbeitet. Beispielsweise werden die Studiengänge aus dem Format, welches StudIP liefert, extrahiert: *[Abschluss:Studiengang (Semesterzahl),]** Folgende Operationen werden darauf durchgeführt:

1. Entferne runde Klammern mit ihrem Inhalt

2. Splitte den String an Kommata
3. Entferne den ANfang eines Strings bis zum ersten Doppelpunkt

```

1 class CSVImporter {
2     /* Instanzvariablen */
3
4     public function __construct($csv_file, $env) { /*...*/ }
5
6     /**
7      * Holt eine Zeile als Array
8      */
9     private function get_line() { /*...*/ }
10
11    /**
12     * importiert die daten aus der csv datei
13     * @return array mit zusammenfassungen
14     */
15    public function import() {
16        while($line = $this->get_line()) { //zeilenweise einlesen
17            $args = array('Titel' => $line[0],
18                        'Vorname' => $line[1],
19                        'Nachname' => $line[2],
20                        'Email' => $line[6],
21                        'Studieninfo' => preg_replace('#\(.?\)\#m', '', $line[11]),
22                        //studieninfo wird von runden klammern und ihrem inhalt entfernt
23                        'MatrNr' => $line[10] == '' ? rand(100000,999999) : $line[10]);
24            /* ... versuchen, den login aus der mail adresse zu extrahieren */
25            /* ... wenn email nicht auf uos.de oder uni-osnabrueck.de endet,
26               namen von sonderzeichen befreien und login generieren */
27            /* ... leere werte auffüllen */
28            /* ... passwort verschlüsseln */
29            try {
30                /* ... studiengänge auseinanderpflücken */
31                /* ... Studenten generieren */
32            } catch (Exception $e) {
33                /* ... Fehler abfangen, interpretieren und in das Rückgabearray schreiben
34                */
35            }
36            try {
37                /* ... student in seine studiengänge einschreiben */
38            } catch (Exception $e) {}
39        }
40        /* ... zusammenfassung zurückliefern */
41    }

```

Gruppenverwaltung

Sascha Kolodzey

Konzept:

- Eine Gruppe gehört zu genau einer Vorlesung.
- Gruppen sind von Tutoren, Übungsleiter und Dozenten der jeweiligen Vorlesung einzusehen.
- Grundsätzlich: Die Tabelle "sindIn" verknüpft die Tabelle "Studenten" mit der Tabelle "Gruppen".
- Also: Studenten "sindIn" Gruppen.
- Schlüssel sind: VerID,GrpID,Login.

- Beteiligte Tabellen: Gruppen, sindIn, Studenten, betreuen, halten. (Informationen über Tutoren/Übungsleiter/Studenten).
- Tabelle "sindIn" verwaltet auch die bearbeiteten Übungsblätter des Studenten in einer Gruppe. (Wichtig bei Verschiebungen).

Frontend / Bedienung

Features:






















- Gruppe zu einer Vorlesung anlegen.
- Gruppe löschen/umbenennen.
- Tutor in der jeweiligen Vorlesung einer Gruppe zuweisen bzw. ändern.
- Studenten aus der jeweiligen Vorlesung einer Gruppe hinzufügen bzw. löschen.
- Studenten innerhalb Gruppen einer Vorlesung verschieben (Gruppen zusammenführen).
- Studenten ab einer Übungsblattnummer einer anderen Gruppe zuweisen.
- Anzeigen aller Gruppen einer Vorlesung.
- (Gruppe in eine andere Vorlesung verschieben)

Übersicht aller Gruppen einer Vorlesung

Gruppen

Suche:

[+ Gruppe hinzufügen](#)

Kürzel	Vorlesung	Tutor	Optionen
Gruppe 1	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  
Gruppe 2	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  
Gruppe 4	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  
Gruppe 5	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  
Einzelbewertung Jonathan Archer	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  
Gruppe 3	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  
Einzelbewertung James Tiberius Kirk	Algorithmen, WS 2008	(dakuenne), Kuenne Daniel	  

Algorithmen WS 2008

Eintrag 1 bis 7 von insgesamt 7

Anfang Zurück 1 Vor Ende

Abbildung 2.1

Über das Dropdown-Menu in der Tabelle kann der Benutzer eine seiner Vorlesungen, die er betreut/liest auswählen. Daraufhin werden alle Gruppen zu dieser Vorlesung geladen und in der Tabelle angezeigt. Die drei Symbole auf der rechten Seite dienen zur Informationsanzeige, Bearbeitung und Löschung der jeweiligen Gruppe. Über den Link "Gruppe hinzufügen", lässt sich eine neue Gruppe in ausgewählte Veranstaltung eintragen.

Gruppe zu einer Vorlesung anlegen.

Gruppe hinzufügen

Vorlesung:

Tutor:

Kürzel:

Abbildung 2.2

Klickt der Benutzer auf den Link "Gruppe hinzufügen" wird er zu dem obigen Formfeld geleitet. Hier ist die ausgewählte Vorlesung bereits selektiert und alle verfügbaren Tutoren zu dieser in das DropDown-Menu "Tutoren" geladen. Wird hier eine andere Vorlesung gewählt, werden alle Tutoren erneut geladen und in dem Menu zur Verfügung gestellt. Das Gruppenkürzel ist frei wählbar und soll der Gruppe einen "Namen" geben.

Gruppe bearbeiten

Gruppe ändern

Vorlesung:

Tutor:

Kürzel:

Abbildung 2.3

Über das Symbol "Bearbeiten" ist es möglich vorhandene Gruppen zu bearbeiten. Wobei es möglich ist den Tutor, das Kürzel und die Veranstaltung zu ändern. Die letztere Operation wird zwar technisch unterstützt, wird aber wahrscheinlich sehr selten Anwendung finden.

Detailansicht einer Gruppe

Gruppenübersicht

Kuerzel: **Gruppe 1** Vorlesung: Algorithmen WS 2008

Hinweis: Gruppe ist voll. Es können keine weiteren Studenten hinzugefügt werden.

Suche:

Student	Studiengang	Von	Bis	
<input type="checkbox"/> (7of9), Of Nine Seven	Menschlichkeit	1	2	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<input type="checkbox"/> (bsisko), Sisko Benjamin	Subraumtheorie;Navigation;Transportertheorie	1	3	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>

Eintrag 1 bis 2 von insgesamt 2 1

Abbildung 2.4

Die Detailansicht einer Gruppe stellt alle wichtigen Informationen einer Gruppe da. Gruppenname und zugehörige Vorlesung werden genannt, sowie ein "Hinweis" Feld, welches den Aktuellen Status einer Gruppe anzeigt.

Es sind drei verschiedene Stadien möglich:

- Status 1: Die Anzahl der Studenten einer Gruppe genügt nicht der Mindestanzahl der dazugehörigen Vorlesung.

Student(en) hinzufügen

Hinweis: Minimale Gruppengröße nicht erreicht (1), Maximal (2).

Abbildung 2.5

- Status 2: Die Anzahl der Studenten ist größer als die Mindestanzahl und kleiner als die Maximalanzahl der dazugehörigen Vorlesung.

Student(en) hinzufügen

Hinweis: Es befinden sich 1 von 2 möglichen Studenten in der Gruppe.

Abbildung 2.6

- Status 3: Die Anzahl der Studenten ist gleich der Maximalanzahl der erlaubten Studenten zu dieser Gruppe.

Siehe Abbildung 2.4

Dessweiteren ist es möglich über die Options Symbole einen Studenten aus der Gruppe zu entfernen, Details zu bearbeiten und sich das Profil des Studenten anzeigen zu lassen.

Student zu einer Gruppe hinzufügen



Abbildung 2.7

Ist die Maximalanzahl von Studenten einer Gruppe nicht erreicht, wird der "Student Hinzufügen"-Link eingblendet, welcher es erlaubt Studenten der Gruppe hinzuzufügen. Das darauf folgende Formular enthält eine Dropdown-Menu welches alle Studenten der jeweiligen Vorlesung anzeigt, die noch nicht in einer Gruppe eingetragen sind. Der Benutzer kann hier auch eine Mehrfachauswahl treffen und somit mehrere Studenten auf einmal der Gruppe hinzuzufügen. Hierbei wird beachtet, dass die Maximalanzahl nicht überschritten wird.

Details zu einem Studenten ändern.

Abbildung 2.8

Über das "Bearbeiten" Symbol in der Detailansicht einer Gruppe ist es möglich Details zu einem Student in der Gruppe zu bearbeiten. Der Benutzer kann den Studenten in eine andere Gruppe der Vorlesung verschieben, oder die "Von" und "Bis" Werten anpassen. Die "Von" und "Bis" Werte geben darüber Auskunft, welche Übungsblätter der Student in dieser Gruppe bearbeitet hat. Es ist zum Beispiel möglich, dass ein Student ab einem bestimmten Übungsblatt in eine andere Gruppe verschoben wird und von da an die Übungsblätter in dieser Gruppe bearbeitet.

Backend / Code

Die Modelklasse Gruppe.php

Die Modelklasse Gruppe.php enthält sämtliche Low-Level Funktionen wie Einfügen, Editieren, Löschen und Speichern von Gruppenobjekten in die Datenbank. Desweiteren enthält die Klasse eine Vielzahl von Funktionen, die bestimmte Datenbankabfragen erledigen z.B. "Hole alle Gruppen zu einer bestimmten Vorlesung".

Ein Codeauschnitt aus der Klasse Gruppe.php:

```

1 class Gruppe extends Model {
2     /**
3      * Liefert alle Gruppen zu dem übergebenen Tutor und zu
4      * der übergebenen Veranstaltung und zu der übergebenen BlattNr.
5      * @return Array von Gruppenobjekten.
6      */
7     public static function get_gruppen_ver_tut_zet($VerID, $TutorLogin, $BlattNr, $env) {
8         /** SQL-Statement */
9         $sql = 'SELECT * FROM Gruppen g WHERE g.VerID = ? and g.TutorLogin = ? AND g.GrpID in
10        (
11            SELECT s.GrpId FROM sindIn s WHERE s.VerID = g.VerID AND
12            s.von <= ? AND s.bis >= ?)';
13         /**Setze Werte, die ersetzt werden sollen */
14         $values = array($VerID, $TutorLogin, $BlattNr, $BlattNr);
15         /**Führe das SQL-Statements mittels Prepared-Statements aus */
16         $result = $env->verbindung()->execute_query($sql,$values);
17         $gruppen = array();
18         /** Verarbeite das Ergebnis */
19         foreach($result as $gruppe) {
20             /** Erzeuge für jede Zeile eine neues Gruppenobjekt*/
21             $gruppen[] = new Gruppe($gruppe,$env);
22         }
23         /**liefer Ergebnis zurück */

```

```

23     return $gruppen;
24 }
25
26 /**
27  * Liefert alle Gruppen zu der übergebenen Veranstaltung.
28  * @param object $VerID
29  * @return Array von Gruppenobjekten
30  */
31 public static function get_gruppen_ver($VerID, $env) {
32     /**SQL-Statement*/
33     $sql = 'SELECT GrpID as id FROM Gruppen WHERE VerID = ?';
34     $value = array($VerID);
35     /**Ergebnis holen */
36     $result = $env->verbindung()->execute_query($sql, $value);
37     $gruppen = array();
38     foreach($result as $id) {
39         /** Erzeuge für jede Zeile eine neues Gruppenobjekt*/
40         $gruppen[] = Gruppe::get_gruppe($id['id'], $env);
41     }
42     return $gruppen;
43 }
44 }

```

Die Controller Klasse Gruppen_Controller.php

Die Klasse Gruppen_Controller.php verfügt über eine Vielzahl von Methoden, die benötigt werden um die entsprechenden Views mit Daten, die der Controller von den Models bekommt, zu füllen. Außerdem verarbeitet der Controller Benutzerangaben und leitet entsprechende Instruktionen, über Models an die Datenbank weiter.

Ein Codeauschnitt aus der Klasse Gruppen_Controller.php

```

1 class Gruppen_Controller extends Controller {
2     /**
3      * Speichert eine neue Gruppe in die Datenbank
4      */
5     function action_speichern() {
6         try {
7             $tutorlogin = null;
8             $verid = null;
9             $kuerzel = null;
10            /* Benutzereingaben werden geprüft
11             und Variablen gesetzt*/
12            if(empty($_REQUEST['select_ver'])) {
13                throw new TuksException('Bitte Vorlesung wählen.');

```

```
34         /** Fehlerbehandlung, Benutzer bekommt Hinweise auf falsche Angaben */
35         $this->env->templateEngine()->assign('error', $e->getMessage());
36         return $this->action_neu($verid, $tutorlogin , $kuerzel);
37     }
38 }
39 }
```


5. Prüfungsverwaltung

von Katharina Röhr und Dominik Abraham

Inhalt:

- Prüfungsmodell
 - Methoden des Models
- Implementation der Getter- und Setter-Methode
 - Überladener Setter
 - Überladener Getter
- Prüfungsverwaltung
 - Views
 - Verwaltungsoberfläche
 - Prüfungsliste
 - Dateneingaben
 - Die Detailansicht
- Notenschlüssel
- Notenschlüsselcontroller

Prüfungsmodell

Klausuren und Übungsblätter werden gemeinsam in einem Model namens 'Prüfung' verwaltet. Die Unterscheidung zwischen Übungsblättern und Klausuren findet durch ein Attribut 'Typ' statt, das entweder auf 'U' oder 'K' steht. Datenbankseitig ist dieses Attribut ein Enumerator mit diesen beiden Werten. Das Model enthält

- Typ, Veranstaltungsschlüssel und Nummer
- Datum
- eine Aufgabenliste

der Prüfung.

Das Aufgabenmodell ist vom Prinzip her eine Wrapperklasse für eine Zahl, da sie ausser ihrem Primärschlüssel nur die Punktzahl für diese Aufgabe verwaltet. Es enthält die Standardmethoden zum Konstruieren und Verändern. Von der Benutzung ist die meiste Funktionalität über das Prüfungsmodell erreichbar, Zugriffe auf die Datenbank über `insert`, `update` und `delete` werden an alle Aufgaben zu der Prüfung weitergeleitet. Da Teile des Aufgabenschlüssels den Prüfungsschlüssel darstellen, und in der Datenbank `ON DELETE CASCADE` gesetzt wurde, wirken sich die Löschoperationen sogar automatisch auf die Aufgaben aus.

Das Prüfungsmodell ist wie auch die für Aufgaben und Notenschlüssel als kurzlebig und unabhängig von der Datenbank konzipiert, die Models werden aus den Datenbankwerten oder manuell konstruiert und können geändert werden, ohne dass es Auswirkungen auf die Datenbank hat. Die Änderungen an der Datenbank werden erst mit bestimmten Methoden angestoßen. Nach Benutzung werden die Models wieder gelöscht. Will man dann etwas ändern, muss man ein neues Model erstellen, um es wie schon zuvor beschrieben, zu benutzen.

[Nach oben](#)

Methoden des Models:

- Erstellen einer Prüfung aus den Daten
- Sich für eine Prüfung die maximale Aufgabenzahl, für die schon Punkte eingetragen sind, holen lassen
- Eine Prüfung mit dem angegebenen Schlüssel aus der Datenbank holen
- Sich alle Prüfungen in einer bestimmten Veranstaltung holen
 - Oder nur eine spezielle Prüfungsart holen
- Nachfragen, ob eine Prüfung zu einem bestimmten Schlüssel existiert
- Eine Auflistung aller Prüfungen aus der Datenbank holen (wird wohl nicht gebraucht, wenn die Datenbank zu viele Prüfungen enthält)
- Eine neue Aufgabe zu der Prüfung hinzufügen
 - Oder ein Array von Aufgaben
- Datenbankfunktionen:
 - Prüfung in die Datenbank schreiben
 - Daten in der Datenbank aktualisieren
 - Prüfung aus der Datenbank löschen
- Getter und Setter der Klasse sind so überladen,
 - dass man die Aufgabenliste durch das Attribut 'Aufgaben' verändern kann ('Aufgaben' steht für die Anzahl der Aufgaben)
 - Verkleinern des Attributs 'Aufgaben' schneidet die Liste zurecht
 - Vergrößern fügt Dummy-Aufgaben mit 0 Punkten ein
 - dass man sich auch ein Attribute holen kann wie
 - 'fix' gibt die maximale Aufgabennummer zurück, für die schon Punkte eingetragen sind
 - 'Aufgaben', Gebrauch siehe oben
 - 'Punkte' die maximale Punktzahl dieser Prüfung

[Nach oben](#)

Implementation der Getter- und Setter-Methode

Die folgenden Quelltexte sind gekürzt und der Übersichtlichkeit halber in einem lesbareren Code verfasst. Die Funktion des echten Programms ist allerdings die gleiche.

Überladener Setter

Auch der Setter prüft, ob das angegebene Attribut existiert. Wenn es existiert und kein Schlüssel ist, wird es auf den übergebenen Wert gesetzt. Wenn der Name unbekannt war, wird geprüft, ob der Name 'Aufgaben' heißt. Wenn es so ist, wird die Aufgabenliste entsprechend verändert. Ist der übergebene Wert kleiner als die alte Listenlänge, werden entsprechend viele Elemente am Ende der Liste gelöscht, ist er größer, werden Aufgaben mit 0 Punkten angehängt.

```
1 function __set($property, $value) {
2     if (array_key_exists($property, $this->properties)){
3         if (gehört_zum_Schlüssel($property)) fehler("Schlüssel dürfen nicht editiert werden.");
4
5         $this->properties["$property"] = $value;
6     }
7     elseif ($property = 'Aufgaben') {
8         if ($this->fix > $value)
9             fehler("Sie löschen Aufgaben, für die bereits Punkte eingetragen wurden. ");
10
11         // lösche alle Aufgaben mit hoeherer Nummer
```

```

12     $Aufgabenzahl = count($this->Aufgabenliste);
13     for($i = $value; $i < $Aufgabenzahl; $i++){
14         loesche($this->Aufgabenliste[$i]);
15     }
16
17     // Fuege Dummy-Aufgaben ein
18     for($i = $Aufgabenzahl + 1; $i <= $value; $i++){
19         $this->addAufgabe(new Aufgabe(array('AufgNr' => $i, 'Punkte' => 0) , $this->env));
20     }
21 } else {
22     fehler('Pruefung->__set: Attribut '.$property.' existiert nicht!');
23 }
24 }

```

[Nach oben](#)

Überladener Getter

Der Getter prüft zunächst ab, ob sich der übergebene Name als Index im property-Array befindet. Ist dies der Fall, wird es zurückgegeben. Ansonsten wird auf andere Attributnamen getestet, und zwar auf 'Aufgaben', 'Punkte', 'fix'. 'Aufgaben' gibt die Aufgabenzahl zurück, 'Punkte' die kumulierte Punktzahl, 'fix' gibt eine Zahl zurück, an Hand derer man prüfen kann, ob an der Prüfung etwas verändert werden darf. Sie beträgt entweder die maximale Aufgabennummer, für die Punkte eingetragen sind, oder ist gleich -1, falls diese Prüfung nicht die letzte ihres Typs ist. Die Aufgaben dürfen nur oberhalb dieser Grenze verändert werden, und die Prüfung darf nur gelöscht werden, wenn fix genau 0 beträgt.

```

1 function __get($property) {
2     if(array_key_exists($property, $this->properties)) {
3         return $this->properties[$property];
4     }
5     } elseif ($property == 'Aufgaben') { // Pseudoattribut 'Aufgaben'
6         return count($this->Aufgabenliste);
7     }
8     } elseif ($property == 'Punkte') { // Pseudoattribut 'Punkte'
9         $sum = 0;
10        foreach($this->Aufgabenliste as $aufg)
11            $sum += $aufg->Punkte;
12        return $sum;
13    }
14    } elseif ($property == 'fix') { // Pseudoattribut 'fix' um zu wissen,
15        return $this->eintraege; // ob editiert werden darf
16    }
17    } else {
18        throw new TuksException('Key '.$property.' not found!');
19    }
20 }

```

[Nach oben](#)

Prüfungsverwaltung:

Pruefung_Controller.php:

Die Klasse **Pruefung_Controller** erweitert die Klasse **Controller** und verwaltet Prüfungen. Da die Aufgaben nur die Punktzahlen zu speichern haben, ist kein **Aufgaben_Controller** vorhanden, alle Operationen an Aufgaben sind komplett im **Pruefung_Controller** eingebettet. Ausser den `action_`-Methoden hat diese Klasse auch noch einige Methoden, um Wiederholungen von häufigen Quelltextausschnitten zu vermeiden:

- Die Methode `req` liest aus dem `$_REQUEST`-Array eine Variable aus, wenn sie gesetzt ist (siehe Notenschlüsselcontroller).
- Die Methode `getPruefung` liest aus dem `$_REQUEST`-Array den Primärschlüssel einer Prüfung aus und holt diese dann aus der Datenbank.
- Die Methode `VerName` gibt den Namen einer übergebenen Veranstaltung zurück.
- Die Methoden `canedit`, `istutor` geben für die jeweiligen Rechte Editierrecht oder Tutorrecht zurück, ob der eingeloggte User diese Rechte hat.
- Die Methode `get_ergebnisse` holt die Ergebnisse des eingeloggten Users zu einer bestimmten Prüfung aus der Datenbank.
- Die Methode `getTypName` gibt zu den übergebenen Werten 'K' oder 'U' den vollständig ausgeschriebenen Namen 'Klausur' oder 'Übungsblatt' zurück.
- Die Methode `info` zeigt ein Info- oder Fehlerfenster mit einer übergebenen Nachricht an.

Die restlichen Methoden sind 'actions', die durch Formulare oder Menüpunkte aufgerufen werden und entweder nur Formulare befüllen oder Daten empfangen und Änderungen verwalten und die Ergebnisse als html-Formular zurückgeben.

Views:

Die Templates sind so angelegt, dass sie oft über boolsche Variablen steuerbar sind, z.B. um die Möglichkeiten zum Löschen oder Ändern je nach Rechten des Users an- oder auszuschalten. Ausserdem werden meist ganze Objekte an die Variablen zugewiesen, sodass man mehrere Informationen über Klassenattribute erzeugen kann. Dabei werden auch die durch die überladenen `__get()`- und `__set()` Methoden realisierten Pseudoattribute benutzt, um häufig benutzte Berechnungen an die Models auszulagern.

Beispiel aus 'pruefung_show.tpl': Wenn jemand die Berechtigung zum Löschen hat, wird ein Löschbutton ('delete.png') angezeigt. Wenn die Prüfung nicht gelöscht werden darf (überprüft durch das Pseudoattribut `fix`), ist der Button grau eingefärbt ('delete_disabled.png'). Die Linkadresse für den Löschbutton wird mit Informationen aus dem Prüfungsobjekt vervollständigt.

```

1 {if ($darf_loeschen)}
2   {if ($pruefung->fix == 0)}
3     <a
4       href="index.php?controller=pruefung&action=loeschen&typ={ $pruefung->Typ}&verid={ $pruefung->Ver.ID}&nr={ $pruefung->Nr}">
5         
6       </a>
7     {else}
8       
9     {/if}
  {/if}

```

Für das Anlegen und Editieren der Daten stehen dabei jeweils 2 actions bereit, die erste gibt ein Formular zum Eingeben der Daten zurück, die zweite wird von dem Formular aufgerufen und setzt die Änderungen oder neuen Prüfungen in der Datenbank fest. Anschließend wird ein html-konformes Informationsfenster erzeugt und daran eine Ansicht der Daten angehängt. Beides wird zurückgegeben, sodass das Infenster über der aktuellen Ansicht erscheint.

Ist beim Ändern der Daten ein Fehler aufgetreten, etwa fehlerhafte Werte, wird anstatt des Informationsfensters ein Fehlerfenster erzeugt und statt der Ansicht erneut das Änderungsformular angezeigt.

Man kann einerseits die Daten der Prüfung, wie etwa das Datum oder die Aufgabenzahl, oder auch die Punktzahlen der Aufgabenliste ändern. Beides sind getrennte Formulare, da es möglich ist, für eine Prüfung, für die schon Punkte eingetragen wurden, die Daten im Nachhinein zu ändern, die Aufgabenliste kann jedoch nur an den Aufgaben verändert werden, für die noch keine Punkte eingetragen wurden.

Es ist nur möglich, Prüfungen zu löschen, wenn keine nachfolgende Prüfung existiert, oder wenn bereits Punkte für diese Prüfung eingetragen wurden.

[Nach oben](#)

Verwaltungsoberfläche

Die Prüfungsverwaltung besteht aus mehreren Oberflächen, von denen aus man die Eigenschaften der Prüfungen kontrollieren und bearbeiten kann. Ausgangspunkt für die Verwaltung ist die Liste aller Prüfungen zu einer Veranstaltung.

Prüfungsliste:

Dazu muss man die Veranstaltung im Seitenmenü ausgewählt haben. Die Liste zeigt die Klausuren und Übungsblätter einer Vorlesung getrennt an und informiert über die Nummer der Prüfung, das Datum, Aufgabenzahl und die maximale Punktzahl der Prüfung an.

Unter den Optionen kann man sich eine Detailansicht der Prüfung zeigen lassen. Für Dozenten und Übungsleiter stehen zusätzlich noch Möglichkeiten zur Verfügung, Prüfungen zu bearbeiten oder zu löschen. Ein Link unter der Auflistung führt zu dem Erstellen von neuen Klausuren/Übungsblättern.

[Nach oben](#)

Dateneingaben:

Für das Erstellen von Prüfungen, das Bearbeiten und das Setzen der Aufgabenlisten gibt es mehrere Formulare, in denen man die Daten zum Setzen und Bearbeiten eingeben kann. Über Buttons am unteren Ende kann man die Daten absenden bzw. das Formular auf Standardwerte zurücksetzen. Kleine grafische Buttons führen aus den Formularen heraus. Ein kleiner grüner Pfeil nach links bedeutet dabei, dass man zurück zur Prüfungsliste gelangt. Ein kleines blaues 'i' führt zur Detailansicht. In beiden Fällen werden eingegebene Daten nicht übernommen.

Natürlich ist es nicht möglich, wesentliche Informationen zu einer Prüfung zu ändern, wie etwa den Typ, die zugehörige Veranstaltung oder die Prüfungsnummer. Änderungen an diesen Daten würden zu Inkonsistenzen in der Datenbank (zum Beispiel doppelte Prüfungsnummern) und zu unvorhersagbarem Verhalten von TuKS führen. Bei der Eingabe eines Datums kommen als EyeSugar Textfelder zum Einsatz, die beim Versuch, sie zu beschreiben, einen Kalender oder ein Feld mit Uhrzeiten aufklappen, über die Datum und Uhrzeit ausgehört werden müssen. Zu beachten ist hierbei, dass die Uhrzeit mit **am** bzw. **pm** versehen ist, und "12:00 am" Mitternacht und "12:00 pm" Mittag bedeutet.

Fehlerhafte Eingaben werden in einem Fehlerfenster gemeldet und sorgen dafür, dass man das Formular erneut ausfüllen muss.

[Nach oben](#)

Die Detailansicht:

In der Detailansicht werden alle Daten einer Klausur oder eines Übungsblattes übersichtlich dargestellt. Die Aufgabenliste ist komplett mit Aufgabennummern und maximal erreichbaren Punktzahlen angezeigt. Ein kleiner grüner Pfeil am unteren Ende der Ansicht führt zurück zu einer Liste aller Übungsblätter und Klausuren dieser Veranstaltung.

Wer Berechtigung zum Ändern einer Prüfung hat, also ein Übungsleiter oder Dozent ist, findet in der Detailansicht zusätzlich noch grafische Buttons zum Bearbeiten oder Löschen einer Prüfung. Insgesamt kommen an drei Stellen neue Buttons hinzu (siehe rote Rahmen): Rechts neben der Aufgabenliste ein Editiersymbol, um die Aufgabenliste zu bearbeiten, unter der Ansicht statt des "Zurück zur Liste"-Buttons ein kleines Menü und darunter ein Link, um Punkte einzutragen, falls man die Berechtigung dafür hat.

Klausur Nr. 2 für 'Algorithmen'

Klausurdetails:

Veranstaltung	Klausurnummer	Aufgabenzahl	Punkte	Abgabedatum
Algorithmen	2	4	100	2009-08-14 11:00:00

Aufgabenliste:

Aufgabe:	1	2	3	4
Punkte:	15	25	25	35



Abb. 1: Detailansicht einer Klausur, für die der der eingeloggte User Berechtigung zum Ändern hat.

Im Menü hat man vier Menüpunkte zur Auswahl:

- Ein grüner Pfeil: Zurück zur Aufgabenliste
- Ein Stift auf einem Blatt Papier: Die gesamte Prüfung bearbeiten
- Ein Notenschlüssel: Notenschlüssel ansehen
(dieser Punkt ist nur bei Klausuren vorhanden)
- Ein Kreuz auf rotem Grund: Die Prüfung löschen



Abb. 2: Menü zum Bearbeiten einer Prüfung

Wer bereits an einer Klausur teilgenommen hat, findet nach dem Eintragen der Punkte eine Detailansicht vor, die ihm zusätzlich zu den üblichen Informationen seine Ergebnisse an Hand einzelner Punktzahlen und die

Note darstellt. Diese Zusatzansicht wird durch die Methode `action_anhang` im `Notenschlüssel_Controller` erstellt.

Klausur Nr. 1 für 'Algorithmen'

Klausurdetails:

Veranstaltung	Klausurnummer	Aufgabenzahl	Punkte	Abgabedatum
Algorithmen	1	4	100	2009-08-14 12:00:00

Aufgabenliste:

Aufgabe:	1	2	3	4
Punkte:	25	25	25	25
Deine Punkte:	20	20	12	13

Notenschlüssel für diese Prüfung:

Note:	1.0	1.3	1.7	2.0	2.3	2.7	3.0	3.3	3.7	4.0	5.0
Punktgrenzen:	100 - 92	91 - 85	84 - 81	80 - 77	76 - 73	72 - 69	68 - 65	64 - 61	60 - 57	56 - 50	49 - 0

Deine Punkte: 65, Deine Note: 3

Abb. 3: Detailansicht einer Klausur, in der der eingeloggte User Punkte hat.

[Nach oben](#)

Notenschlüssel

Der Notenschlüssel einer Klausur trägt die Note (von 1.0 bis 5.0) gegen die maximale Punktzahl auf, für die man noch diese Note erhalten würde. Durch ihn lässt sich aus den eingetragenen Ergebnissen automatisch eine Note berechnen.

Das Model zum Notenschlüssel kann aus einer Prüfung einen Standardnotenschlüssel berechnen. Die zugehörige Prüfung wird dem Konstruktor übergeben.

```

1 // Array mit Standardprozentzahlen
2 // da php keine floats als Arrayindices erlaubt,
3 // werden die Noten * 10 als Key genommen
4
5 private $std_steps = array(10 => 100,
6                             13 => 91,
7                             17 => 84,
8                             20 => 80,
9                             23 => 76,
10                            27 => 72,
11                            30 => 68,
12                            33 => 64,
13                            37 => 60,
14                            40 => 56,
15                            50 => 49);
16 protected $steps = array();
17
18 function __construct(Pruefung $pruefung, $env) {
19     parent::__construct($env);
20
21     $Punktzahl = $pruefung->Punkte;
22     foreach($this->std_steps as $note => $punkte){
23         $this->steps[$note] = round(($punkte * $Punktzahl) / 100);
24     }
25     sortieren_nach_Schlüssel($this->steps);
26 }

```

Daneben stellt das Model noch andere Methoden zur Verfügung, u.a.:

- `function exists($VerID, $Nr, $env)`: Prüft, ob es zu dem angegebenen Primärschlüssel einen Notenschlüssel in der Datenbank gibt.

- `function insert()`: Schreibt den Notenschlüssel in die Datenbank
- `function update()`: Aktualisiert den Notenschlüssel in der Datenbank
- `function check()`: Prüft den Notenschlüssel auf Konsistenz und gibt einen entsprechenden Wahrheitswert zurück. Es wird geprüft, ob alle Punktzahlen zwischen 0 und der maximalen Punktzahl der Prüfung sind, und ob wirklich die Punktzahlen für bessere Noten monoton ansteigen.
- `function __get($property)` ist wieder so überladen, dass man auf zusätzliche Pseudoattribute zugreifen kann.
 - `$notenschluessel->Note23` gibt beispielsweise die Punktzahl für die Note 2,3 zurück.
 - `$notenschluessel->Array` gibt ein Array der Bauart `{"1.0" => xy, "1.3" => xyz}` zurück, wobei jeweils die Note als String auf die maximale Punktzahl für diese Note abgebildet wird.

[Nach oben](#)

Notenschlüsselcontroller

Ein paar Methoden zur Übersicht

- Die `function __construct($env)` ist der Konstruktor der Klasse `Notenschluessel_Controller`
- Die `function req($attr, $default = false)` gibt eine Variable aus dem Request-Array zurück, sonst ein Defaultwert
- Die `function getNotenschluessel()` gibt aus dem Schlüssel, der per `$_REQUEST` übergeben wurde, den passenden Notenschlüssel zurück
- Die `function action_speichern()` legt eine neue Prüfung an und schreibt sie in die Datenbank
- Die `function action_zuruecksetzen()` ersetzt den Notenschlüssel der Klausur durch einen automatisch erzeugten Standard-Notenschlüssel. Dies ist mit einem Funktionsaufruf der Prüfungsklasse getan.

```
1 $pr->stdNotenschluessel(true); // der übergebene Wert 'wahr' bedeutet, alte Notenschlüssel
   werden überschrieben
```

- Die `function action_anhang($ergebnisse)` generiert einen Anhang, um unter einer Prüfung den Notenschlüssel samt Ergebnissen anzuzeigen. Die Ergebnisse werden aufsummiert und als Gesamtpunktzahl und Endnote dargestellt.

```
1 if ($ergebnisse) {
2     $punkte = array_sum($ergebnisse);
3     $this->env->templateEngine()->assign('allePunkte', $punkte);
4     $this->env->templateEngine()->assign('endnote', $Notenschluessel->getNote($punkte));
5 }
6 return $this->env->templateEngine()->fetch('notenschluessel_anzeigen.tpl');
```

- Die `function canedit` gibt an, ob der Benutzer berechtigt ist, Notenschlüssel anzulegen oder zu ändern

```
1 function canedit(){
2     return anzahl_elemente(
3         schnittmenge(
4             $this->meine_rollen(),
5             $this->editors)
6         ) != 0;
7 }
```


Der Notenschlüsselcontroller implementiert die Funktionalitäten zum Ansehen und Ändern von Notenschlüsseln, analog zu den Prüfungen. Die Formulare haben dabei das gleiche Aussehen, die Erfahrungen mit der Prüfungsverwaltung können also direkt auf die Notenschlüsselverwaltung übertragen werden.

[Nach oben](#)

6. Klausurpunkte eintragen

von Anita Brankova und Antje Siemer

Inhalt:

- Oberfläche
- Funktionalität
- Implementierte Controller
- Implementierte Views

Oberfläche

Wenn man im TUKS-Menue eine bestimmte Veranstaltung ausgewählt hat, gibt es die Möglichkeit, die Punkte für Klausuren in dieser Veranstaltung einzutragen. Wenn man sich für *Endnote eintragen* entschieden hat, erscheint eine Liste aller in dieser Veranstaltung geschriebenen Klausuren. Nach Auswahl einer Klausur erscheint ein Formular, in dem für die Punkteeingabe ein Inputfeld pro Aufgabe und pro zugelassenen Studenten vorhanden ist. Falls für diese Klausur schon für einige, oder auch alle, teilgenommenen Studenten Punkte eingetragen wurden, werden sie in diesem Formular mit ausgegeben und können nun geändert werden. Für die nicht zugelassenen Studenten gibt es die Möglichkeit, sie über ein AutoComplete-Feld hinzuzufügen, sodass auch hier ggf Punkte eingetragen werden können. Auch hier werden bereits eingetragene Punkte schon mit ausgegeben. Der Student muss in diesem Fall also auch nicht erneut manuell hinzugefügt werden. Da es auch vorkommen kann, dass Besucher der Vorlesung zwar die Zulassung zur Klausur bekommen haben, aber dennoch nicht teilnehmen, gibt es in diesem Formular eine Checkbox *nicht angemeldet*. Falls diese gesetzt ist werden für den entsprechenden Studenten keine Werte in der Datenbank gespeichert.

Klausuren von "Algorithmen (WS 2008)"

Klausurnummer	Datum
Klausur1	2009-08-14 12:00:00
Klausur2	2009-08-14 11:00:00

Abb. 17: Liste aller in dieser Veranstaltung geschriebenen Klausuren

Klausur 1
Formular für Punkteeingabe

mit Zulassung:
Suche:

Student:	Nr. 1	Nr. 2	Nr. 3	Nr. 4	nicht angemeldet
Charles Tucker (MatrNr.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Jean-Luc Picard (MatrNr.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Seven Of Nine (MatrNr.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Eintrag 1 bis 3 von insgesamt 3 1

ohne Zulassung:

Student:	Nr. 1	Nr. 2	Nr. 3	Nr. 4	nicht angemeldet
Zephram Cochrane (MatrNr.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Terence Hill (MatrNr.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Kathryn Janeway (MatrNr.)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Abb. 18: Formular zum Eintragen der Punkte pro Aufgabe

Funktionalität

- Die Punkte für eine Klausur dürfen nur die jeweiligen Dozenten und Übungsleiter eintragen bzw ändern.
- Für die Realisierung der unter *Oberfläche* beschriebenen Funktionalität sind zwei Aktionen und Views notwendig:

action_list_klausur()

In dieser Aktion werden alle zu der aktuellen Veranstaltung gehörenden Prüfungen mit dem Prüfungstyp *K* (Klausur) aus der Datenbank geholt und dann dem Template *klausur_list.tpl* übergeben, das dann schließlich alle Klausuren auflistet.

action_punkte_eintragen()

Hier werden zunächst alle Besucher der aktuellen Veranstaltung aus der Datenbank geholt, welche dann weiter über eine Methode aus *Statistiken*, welche die Zulassungen für einzelne Studenten berechnet, in Teilnehmer mit und ohne Zulassung aufgespalten wird. In einer statischen Hilfsmethode *get_studenten_punkte(\$verid, \$klausur_nr, \$aufgaben, \$teilnehmer,\$env)* (siehe unten) werden dann die bereits bestehenden Ergebnisse aus der Datenbank geholt. Für die zugelassenen Studenten werden alle benötigten Daten dem Template *punkteKl_eintragen.tpl* übergeben. Im Gegensatz dazu wird für die Studenten ohne Zulassung speziell geprüft, ob Ergebnisse in der Datenbank vorhanden sind und nur in diesem Fall werden die Daten dem Template übergeben.

```

1 foreach($teilnehmer_oz as $teiln) {
2   if(in_array($teiln['Login'],array_keys($punkte))) { // $punkte: bereits bestehende
3     $values2[$teiln['Login']] = array('vorname' => $teiln['Vorname'],
4                                     'nachname' => $teiln['Nachname'],
5                                     'login' => $teiln['Login']);
6   }
7 }

```

Für die Verarbeitung/Speicherung der neu eingefügten Daten wird zunächst wieder getestet, ob dazu schon Daten in der Datenbank existieren. Falls dies der Fall ist, wird lediglich ein *UPDATE* in der Datenbank durchgeführt. Andernfalls werden die Daten neu eingefügt.

benötigte Hilfsmethoden:

get_studenten_punkte(\$verid, \$klausur_nr, \$aufgaben, \$teilnehmer,\$env)

```

1 static function get_studenten_punkte($verid, $klausur_nr, $aufgaben, $teilnehmer,$env) {
2     $sql ="SELECT Login, AufgNr, Punkte
3         FROM bearbeitenKl
4         WHERE VerID = " . $verid . "
5         AND PrfID = " . $klausur_nr;
6
7     $ergebnisse = $result=$env->verbindung()->execute_query($sql);
8
9     $ergebnis_2 = array();
10
11     foreach($teilnehmer AS $item) {
12         foreach($aufgaben AS $key => $aufgabe) {
13             foreach($ergebnisse AS $ergebnis) {
14                 if(in_array($item['Login'], $ergebnis) && in_array($aufgabe->AufgNr, $ergebnis))
15                 {
16                     $aufg_punkte[$aufgabe->AufgNr - 1] = $ergebnis['Punkte'];
17                     $ergebnis_2[$item['Login']] = $aufg_punkte;
18                 }
19             }
20         }
21     }
22     return $ergebnis_2;
23 }

```

Mit dieser Methode werden zunächst alle bereits bestehenden Ergebnisse aus der Datenbank geholt. Dann wird für jeden Studenten und jede Aufgabe einer bestimmten Klausur in einer Veranstaltung getestet, ob diese Kombination im Ergebnis vorhanden ist. Des weiteren wird dann in einem Array für jeden Studenten jede Aufgabe das jeweilige Ergebnis gespeichert. Falls keine Werte vorhanden sind, wird an dieser Stelle im Array NULL gespeichert.

Implementierte Controller:

- Methoden im Klausurerg_Controller.php:
 - *action_list_klausur()*:
 - *action_punkte_eintragen()*:

Implementierte Views:

- *action_list_klausur()*: es werden alle zu der aktuellen Veranstaltung gehörenden Prüfungen aus der Datenbank geholt
- *punkteKl_eintragen.tpl*: gibt die Studenten mit Zulassung und ohne Zulassung und ihren Aufgaben in einer Tabelle aus

[Nach oben](#)

7. Veranstaltungsverwaltung

von Anita Brankova und Antje Siemer

Inhalt:

- Einführung
- Aktion
- Models
 - Veranstaltung
 - Vorlesung
 - Seminar
- Controller
- Views
- Aktion `veranstaltung_edit()`

Einführung

Die Klasse `Veranstaltung` ist die Oberklasse der Klassen `Seminar` und `Vorlesung`. Es handelt sich hierbei um eine vollständige Spezialisierung, d.h. eine `Veranstaltung` muss entweder `Seminar` oder `Vorlesung` sein. Eine `Veranstaltung` verfügt über die Attribute `VeranstaltungsID`, `Titel`, `Semestertyp`, `Semesterjahr` und die maximale Anzahl an teilnehmenden Studenten. Bei der `VeranstaltungsID` handelt es sich um eine künstlich eingeführte Nummer, über die die jeweilige `Veranstaltung` eindeutig identifiziert werden kann. Diese ID ist also der Primärschlüssel einer `Veranstaltung`. Außerdem ist das Attribut `Titel` in Kombination mit `Semestertyp` und `Semesterjahr` *unique*, d.h. 'dieselbe' `Veranstaltung` darf nicht mit unterschiedlicher `VeranstaltungsID` mehrfach in der Datenbank eingefügt werden.

Die Klasse `Vorlesung` besitzt zusätzlich zu den Attributen einer `Veranstaltung` noch folgende bei der Bewertung der `Übungszettel` zusammenhängende Attribute: einen Enum-Wert `proZettel`, der angibt, ob der angegebene Prozentsatz pro `Zettel` oder über die Summe aller `Zettel` erreicht werden muss, einen Prozentsatz, die Anzahl der `Freiversuche`, jeweils eine Angabe zur minimalen und maximalen Gruppengröße und die Gesamtanzahl der `Übungszettel`.

Die Klasse `Seminar` besitzt keine zusätzlichen Attribute zur `Veranstaltung`.

Aktionen

- Für die `Veranstaltungsverwaltung` sind folgende Aktionen notwendig:
 - Anzeigen von einzelne `Veranstaltungen` bzw Ausgabe einer Liste aller `Veranstaltungen`
 - eine `Veranstaltung` muss neu angelegt werden können (muss aber dabei als `Seminar` oder `Vorlesung` gekennzeichnet werden)
 - Änderung von bestehenden `Veranstaltungen`
 - Tutoren/`Übungsleiter` müssen den entsprechenden `Veranstaltungen` zugeordnet werden bzw sie müssen auch auch wieder geändert oder ausgetragen werden können
 - Löschen von bestehenden `Veranstaltungen`

Models

Veranstaltung

- Die Klasse Veranstaltung erbt von der abstrakten Klasse 'Model'.
- Alle Attribute aus der Datenbank werden in einem assoziativen Array *properties* gespeichert. Die VeranstaltungsID ist hierbei allerdings eine Ausnahme, sie wird in einem extra Array *keys* gespeichert.
 - Die Attribute werden getrennt, da die VeranstaltungsID nicht manuell gesetzt werden soll, sondern in der Datenbank durch Auto-Inkrement ermittelt wird.
- Im Konstruktor werden die übergebenen Attribute zunächst im entsprechenden Array gespeichert.

Zur Umsetzung der Aktionen werden folgende Funktionen benötigt:

statische Funktionen:

- `get_veranstaltungen($env)`: liefert alle in der Datenbank enthaltenen Veranstaltungen zurück
- `get_veranstaltung($verid, $env)`: liefert die zur VeranstaltungsID zugehörige Veranstaltung zurück
- `get_some_veranstaltungen($ver_ids, $env)`: liefert bei Übergabe eines Arrays von VeranstaltungsIDs die Liste der ausgewählten Veranstaltungen zurück
- `rolle($ver_id, $login, $env)`: gibt bei Übergabe einer VeranstaltungsID und eines Logins die 'Rolle' der Person in der ausgewählten Veranstaltung zurück
 - Bsp: Unterscheidung zwischen *Tutor* und *Tutor und Student*

```

1  ...else {
2    $query="SELECT * FROM betreuen AS b
3        WHERE b.VerID= ".$ver_id."
4        AND b.Login='".$login.'"
5        AND '".$login.'" NOT IN
6            (SELECT Login from besuchen AS be
7              WHERE be.VerID=".$ver_id.");
8    $result = $env->verbindung()->execute_query($query);
9
10   if(count($result) > 0) {
11       $rolle[]='Tutor';
12       return $rolle;
13   }
14   else{$query="SELECT * FROM betreuen AS b
15       WHERE b.VerID= ".$ver_id."
16       AND b.Login='".$login.'"
17       AND '".$login.'" IN
18           (SELECT Login FROM besuchen AS be
19             WHERE be.VerID=".$ver_id.");
20       $result = $env->verbindung()->execute_query($query);
21   }
22   if(count($result) > 0){
23       $rolle[]='Tutor';$rolle[]='Student';
24   }
25   else {...

```

- `exists_static($verid, $env)`: liefert einen Boolean-Wert zurück, je nachdem ob die übergebene ID in der Datenbank vorhanden ist
- `gleiche_ver($titel, $semestertyp, $semesterjahr, $env)`: testet, ob Titel und Semestertyp und -jahr schon in der Form in der Datenbank existieren
- `besuchen($ver_id, $env)`: liefert ein Array aller Studenten zurück, die diese Veranstaltung besuchen
- `update($verid, $titel, $semestertyp, $semesterjahr, $maxstudenten, $env)`: ändert die Werte einer Veranstaltung
- `typ($verid, $env)`: gibt für die Veranstaltung zurück, ob es sich um eine Vorlesung oder ein Seminar handelt

- Bsp:Vorlesung

```

1  ..if(Veranstaltung::exists_static($verid, $env)){
2      $sql="SELECT COUNT(*)AS anzahl FROM Veranstaltungen AS v, Vorlesungen AS vo
3          WHERE v.VerID=vo.VerID
4          AND v.VerID=".$verid;
5      $result=$env->verbindung()->execute_query($sql);
6          if ($result[0][['anzahl']]>0) {
7              $typ['Typ']='Vorlesung';
8          }..

```

- test_max(\$maxstud): überprüft, ob die eingegebene Anzahl von Studenten zulässig ist (Bsp:Es dürfen keine negativen Werte eingegeben werden)

nicht-statische Funktionen:

- __get(\$property): sucht in den beiden assoziativen Arrays 'properties' und 'keys' nach der übergebenen Variable und gibt dann den passenden Wert zurück, falls die Variable in einem der beiden Arrays vorhanden ist
- __set(\$property,\$value): sucht analog zu __get(\$property) die Variable und setzt dann den entsprechenden Wert
- insert(): Falls die Veranstaltung noch nicht vorhanden ist(siehe exists_ver()) und die eingegebenen Werte zulässig sind(siehe test_ver()), wird sie in die Datenbank eingefügt. Die VeranstaltungsID zu dieser Veranstaltung wird dadurch ermittelt, dass die höchste ID, also die der zuletzt eingefügten Veranstaltung, aus der Datenbank geholt wird.
- exists_v(): liefert einen Boolean-Wert zurück, je nachdem ob die ID der aktuellen Veranstaltung in der Datenbank vorhanden ist
- exists_ver(): testet, ob Titel und Semestertyp und -jahr der aktuellen Veranstaltung schon in der Form in der Datenbank existieren
- delete(): löscht die aktuelle Veranstaltung
- test_ver(): überprüft, ob die eingegebene Anzahl von Studenten zulässig ist

Vorlesung

- Die Klasse Vorlesung erweitert die Klasse Veranstaltung.
- Die zusätzlich Attribute werden in einem Array *v_properties* gespeichert.
- Im Konstruktor wird das Array der übergebenen Argumente zunächst aufgeteilt. Je nachdem, ob sich die Variable im Array 'keys','properties' oder 'v_properties' befindet, werden die Werte in unterschiedlich Arrays (\$args_veranstaltung,\$args_vorlesung) gespeichert.Die VeranstaltungsID wird direkt gesetzt. Des Weiteren wird dem Konstruktor der Oberklasse das Array '\$args_veranstaltung'übergeben. Die Attribute aus '\$args_vorlesung' werden hier im Vorlesungs-Konstruktor gesetzt. Falls die Veranstaltung in der Datenbank noch nicht vorhanden ist, wird im Konstruktor die Funktion *insert()* aufgerufen.

Zur Umsetzung der Aktionen werden folgende Funktionen benötigt:

statische Funktionen:

- get_vorlesung(\$verid, \$env): liefert bei Übergabe einer VeranstaltungsID die entsprechende Vorlesung zurück. (Es werden die Attribute einer Vorlesung *und* einer Veranstaltung aus der Datenbank geholt.)
- get_some_vorlesungen(\$verids, \$env): gibt eine Liste der ausgewählten Vorlesungen zurück
- get_vorlesungen(\$env): gibt alle in der Datenbank enthaltenen Vorlesungen zurück
- vorlesung_update(\$verid, \$titel, \$semestertyp, \$semesterjahr, \$maxstudeten, \$prozettel, \$prozentsatz, \$anzfreiversuche, \$mingruppe, \$maxgruppe, \$anzueb, \$env): Es wird zunächst mit den entsprechenden Attributen die Methode *update(...)* aus der Oberklasse aufgerufen. Mit den übrigen Attributen wird ein *UPDATE* in 'Vorlesungen' in der Datenbank durchgeführt.

- `test_static($prozs,$anzfr,$anzueb,$mingruppe,$maxgruppe)`: Es wird geprüft, ob die eingegebenen Werte zulässig sind, beispielsweise darf die Anzahl der Freiversuche nicht größer sein als die Anzahl der Übungsblätter.

nicht-statische Funktionen:

- `__get($property)`: sucht in den drei assoziativen Arrays 'v_properties', 'properties' und 'keys' nach der übergebenen Variable und gibt dann den passenden Wert zurück, falls die Variable in einem der drei Arrays vorhanden ist
- `__set($property,$value)`: sucht analog zu `__get($property)` die Variable und setzt dann den entsprechenden Wert
- `vorlesung_insert()`: In dieser Methode wird zunächst die `insert()`-Funktion aus der Oberklasse aufgerufen, d.h. es wird erst eine Veranstaltung in die Datenbank eingefügt. Da in dieser Methode auch die VeranstaltungsID der neu eingefügten Veranstaltung aus der Datenbank geholt wird, kann nun in `vorlesung_insert()` eine Vorlesung mit der passenden ID eingefügt werden.
- `delete_vorlesung()`: Es wird erst die `delete()`-Methode aus *Veranstaltung* ausgeführt, dann wird in `delete_vorlesung` die aktuelle Vorlesung aus der Tabelle *Vorlesungen* in der Datenbank gelöscht.
- `test()`: Es werden alle in die Datenbank einzufügenden Werte auf Korrektheit überprüft, d.h. es dürfen zum Beispiel keine negativen Werte eingefügt werden. (analog zu `test_static`)

Seminar

- Die Klasse Seminar erweitert die Klasse Veranstaltung.
- Da ein Seminar aber keine zusätzlichen Attribute zu einer Veranstaltung besitzt wird im Konstruktor lediglich der Konstruktor der Oberklasse aufgerufen.

Zur Umsetzung der Aktionen werden folgende Funktionen benötigt:

statische Funktionen:

- `get_seminar($verid, $env)`: Es wird das ausgewählte Seminar zurückgeliefert.
- `get_some_seminare($verids, $env)`: In dieser Methode wird eine Liste der ausgewählten Seminare ausgegeben.
- `get_seminare($env)`: Es wird eine Liste aller in der Datenbank enthaltenen Seminare zurückgegeben.

nicht-statische Funktionen:

- `__get($property)`: Wie schon in *Veranstaltung* wird auch hier in den beiden Arrays *properties* und *keys* nach der übergebenen Variable gesucht und dann, falls vorhanden, der passende Wert zurückgegeben.
- `__set($property, $value)`: Es wird analog zu `__get($property)` nach der Variable gesucht und dann der entsprechende Wert gesetzt.
- `insert()`: Falls das Seminar noch nicht vorhanden ist, wird zunächst die `insert()`-Methode aus *Veranstaltung* aufgerufen und dann die entsprechende VeranstaltungsID in die Tabelle *Seminare* in der Datenbank eingefügt.
- `delete_seminar()`: Diese Methode löscht ein Seminar, indem erst die Veranstaltung und dann das Seminar aus der Datenbank gelöscht wird.

Controller

Veranstaltung Controller:

Es besteht die Möglichkeit zwischen folgenden Aktionen zu wählen:

- `action_show()`: zeigt alle Informationen zu einer ausgewählten Veranstaltung
- `action_list()`: zeigt eine Liste mit allen Veranstaltungen
- `action_new()`: legt eine neue Veranstaltung an
- `action_edit()`: bearbeitet eine bestehende Veranstaltung
- `action_delrequest()`: fragt nach, ob eine Veranstaltung wirklich gelöscht werden soll
- `action_delete()`: löscht eine Veranstaltung
- `action_()`: Default-Aktion, die aufgerufen wird, wenn keine Aktion ausgewählt wurde

Views

Für die oben genannten Aktionen werden folgende Views benötigt:

- `veranstaltung_view`: zeigt alle Details einer Veranstaltung
- `veranstaltung_list`: zeigt eine Liste aller Veranstaltungen
- `veranstaltung_insert`: gibt ein Formular zur Eingabe einer Veranstaltung aus
- `veranstaltung_edit`: gibt ein Formular zur Bearbeitung einer Veranstaltung aus

Aktion `veranstaltung_edit()`:

- Für eine bestehende Veranstaltung werden die aktuellen Werte geändert und neu in die Datenbank geschrieben.
- Diese Aktion kann nur von Übungsleitern und Tutoren ausgeführt werden.

Abb. 19: Das Formular für Veranstaltung bearbeiten.

Realisierung:

- Zunächst werden über die aktuelle VeranstaltungsID alle Werte der Veranstaltung aus der Datenbank geholt.
- Es werden zusätzlich die zu dieser Veranstaltung eingetragenen Übungsleiter und Tutoren geholt.

```

1  ...$sql_u="SELECT p.Vorname, p.Nachname, p.Login FROM Personen p,halten b,Angestellte a
2          WHERE p.Login=b.Login
3          AND a.Login=p.Login
4          AND a.Rolle='Uebungsleiter'
5          AND b.VerID=".$_REQUEST['verid'];
6          $result_u=$this->env->verbindung()->execute_query($sql_u);
7          $this->env->templateEngine()->assign('leiter', $result_u);
8
9  $sql_t="SELECT p.Vorname, p.Nachname, p.Login FROM Personen p, betreuen b
10         WHERE p.Login=b.Login AND b.VerID=".$_REQUEST['verid'];
11         $result_t=$this->env->verbindung()->execute_query($sql_t);
12         $this->env->templateEngine()->assign('tutoren', $result_t);
13
14 ...

```

- Im Formular werden die aktuellen Werte der Veranstaltung ausgegeben. Diese können nun in den entsprechenden Feldern geändert und wieder gespeichert werden.
- Bei Tutoren können wie bei *action_new* ggf noch zusätzlich Tutoren hinzugefügt werden.
- Die Tutoren und Übungsleiter werden wie bei *action_new* per Autocomplete in das Formular eingefügt.
- Bei den Veranstaltungsattributen können die Änderungen einfach über ein *update()* in der Datenbank durchgeführt werden. Bei den Tutoren und Übungsleitern ist dies allerdings nicht möglich, da bei einer Änderung Schlüssel betroffen wären. Deshalb werden bei dieser Aktion alle vorhandenen Tutoren und Übungsleiter zunächst gelöscht und erst danach werden die eingetragenen Personen neu in die Datenbank eingetragen.

```
1 //lösche alle Übungsleiter
2 foreach($result_u as $uebl ){
3     $halten=new Halten(array('Verid'=>$_REQUEST['verid'],'Login'=>$uebl['Login']),$this->env);
4     $halten->delete();
5 }
6 //lösche alle Tutoren
7 foreach($result_t as $tutor){
8     $betreuen=new
9     Betreuen(array('VerID'=>$_REQUEST['verid'],'Login'=>$tutor['Login']),$this->env,$insert=FALSE);
10    $betreuen->delete();
11 }
12 //Übungsleiter eintragen
13 for($i = 1; $i <= 2; $i++) {
14     if(!empty($_REQUEST['uebungsleiter_login_.$i']) && (!empty($_REQUEST['uebungsleiter_.$i'])))
15     {
16         $uebungsleiter = new Halten(array("Login" => $_REQUEST['uebungsleiter_login_.$i'], "Verid"
17         => $_REQUEST['verid']), $this->env);
18     }
19 }
20 //Tutoren eintragen
21 for($j=1; $j<=$_REQUEST['tutor_count']; $j++){
22     if((!empty($_REQUEST['tutor_login_.$j']))&& (!empty($_REQUEST['tutor_.$j'])))
23     $tutor = new Betreuen(array("Login" => $_REQUEST['tutor_login_.$j'], "VerID" =>
24     $_REQUEST['verid']), $this->env, $insert = TRUE);
25 ...
26 }
```

[Nach oben](#)

8. Seminarnote

von Sabine Heider, Katja Sperber

Inhalt:

- Oberfläche
- Funktionalität
- Abfangen fehlerhafter Benutzereingaben
- Implementierte Controller
- Implementierte Views

Oberfläche

Es erscheint eine Tabelle, die alle für das Seminar eingetragenen Studenten auflistet und ein Eingabefeld für die Note bereitstellt. Wenn ein Teilnehmer bereits über eine Note verfügt, wird diese im Eingabefeld angezeigt, andernfalls bleibt das Feld leer. Desweiteren existiert eine Spalte Optionen, die über die Buttons 'Informationen anzeigen' und 'Austragen' verfügt.

Seminar [zurück](#)

Übersicht der Teilnehmer

Suche:

MatrNr	Vorname	Nachname	Note	Optionen
111115	Albert	Burkhalter	1.7	 
121117	Andrew	Carter	2.0	 
193547	Zephram	Cochrane	2.3	 
121116	Terence	Hill	5.0	 

Eintrag 1 bis 4 von insgesamt 4 Anfang Zurück 1 Vor Ende

Abb. 20: Formular zur Seminarnoteneingabe

Funktionalität

In den entsprechenden Eingabefeldern kann für jeden Studenten eine Endnote eingetragen werden. Wenn ein Feld leer bleibt, das anfangs mit der in der DB eingetragenen Note belegt war, wird dieser Eintrag aus der DB gelöscht. Nach erfolgreicher Noteneingabe erscheint neben einer Infomeldung erneut die Tabelle mit den Eingabefeldern.

Abfangen fehlerhafter Benutzereingaben

- Noteneingabe mit Kommata wird toleriert:

Wenn eine Note mit einem Komma statt eines Punktes eingefügt wird, wird das Komma mit folgendem Kommando ersetzt

```
1 $note = str_replace(',', '.', $note);
```

- ungültige Note:

In einem array werden alle gültigen Noten eingetragen.

```
1 $noten = array (1.0, 1.3, 1.7, 2.0, 2.3, 2.7, 3.0, 3.3, 3.7, 4.0, 5.0);
```

und mit der übergebenen Note abgeglichen. Alle gültigen eingetragenen Noten werden in jedem Fall übernommen, bei den anderen kommt es zu einer Fehlermeldung und der alte Wert bleibt erhalten.

Implementierte Controller:

- Methoden im Seminar_Controller.php:
 - *action_seminarteilnehmer()*: holt Studenten und aktuelle Noten aus der DB und übergibt sie der View *seminarnote.tpl*
 - *action_seminarnote()*: speichert die eingetragenen Noten in der DB
 - *action_austragenreq()*: ruft das Bestätigungsfenster auf
 - *action_austragen()*: trägt bei Bestätigung einen Studenten aus einem Seminar aus

```
1 $besuchen = Besuchen::get_besuchen ($REQUEST['id'], $_GET['verid'], $this->env);
2 $student = Student::get_student($REQUEST['id'], $this->env);
3 // wenn die Bestätigung durch den User erfolgt, wird gelöscht
4 if (isset($_REQUEST['delete_confirm'])) {
5     $besuchen->delete();
6     $info = $student->Vorname." ".$student->Nachname." wurde erfolgreich aus dem Seminar".
7         " ausgetragen.";
8 }
9 // andernfalls nicht
10 else {
11     $info = $student->Vorname." ".$student->Nachname." wurde nicht aus dem Seminar".
12         " ausgetragen." ;
13 }
```

Implementierte Views:

- *seminarnote.tpl*: gibt übergebene Studenten mit ihren Noten in einer Tabelle aus

9. Punkte für Übungsblätter

von Katja Sperber und Sabine Heider

Inhalt:

- Oberfläche
- Funktionalität
 - Einzelbewertung
- Abfangen fehlerhafter Benutzereingaben
- Implementierte Controller
 - Tutorerg_Controller.php
- Implementierte Views

Oberfläche

Es erscheint eine Tabelle, die für die ausgewählte Veranstaltung alle verfügbaren Übungsblätter auflistet. Nach Auswahl eines der Übungsblätter erscheinen alle Gruppen, die für diese Übungsblatt und die eingeloggte Person eingetragen sind. Zusätzlich wird zu jeder Gruppe die bisher erreichte Gesamtpunktzahl ausgegeben. Im folgenden Formular werden zu der angeklickten Gruppe die entsprechenden Teilnehmer und die Aufgaben mit den bisher eingetragenen Punkten aufgelistet. Unterhalb der Gruppeninfo befindet sich entweder ein Button, der eine Einzelbewertung der Gruppenmitglieder ermöglicht, oder ein Informationsfeld, das besagt, dass keine Einzelbewertung aufgrund der Gruppengröße möglich ist.

Übungsblatt 1 [zurück zu den Gruppen](#)

Gruppeninfo 4

Name	Matrikelnummer
Jean-Luc Picard	121118
Charles Tucker	890084

Formular für Punkteeingabe

Punkte eintragen

Aufgabe 1 (max. Punktzahl: 5):	<input type="text" value="5"/>
Aufgabe 2 (max. Punktzahl: 5):	<input type="text" value="5"/>

Abb. 21: Formular zur Punkteingabe

Funktionalität

Dozenten, Übungsleiter und Tutoren sind in der Lage, Punkte für Übungsblätter einzutragen. Die Funktionalität unterscheidet sich jedoch leicht. Während Dozenten und Übungsleiter alle Gruppen der Veranstaltung bewerten können, dürfen Tutoren dies nur für die ihnen zugewiesenen Gruppen.

vereinfachter Code, der die entsprechenden Gruppen aus der DB holt:

```
1 $halten = Halten::get_halten($args,$this->env);
2 // wenn die eingeloggte Person die Veranstaltung hält, werden alle Gruppen zu der
3 // Veranstaltung und einem bestimmten Zettel ausgegeben
4 if (!is_null($halten)) $gruppen = Gruppe::get_gruppen_ver_zet($verid, $blattnr, $this->env);
5 // andernfalls handelt es sich um einen Tutor, der nur seine eigenen Gruppen bewerten darf
6 else $gruppen = Gruppe::get_gruppen_ver_tut_zet($argv, $this->env);
```

Einzelbewertung

Wenn sich mehr als eine Person in der Gruppe befindet, kann eine Einzelbewertung vorgenommen werden.

vereinfachter Code; überprüft, ob eine Einzelbewertung möglich ist

```
1 // liefert alle Studenten aus der Gruppe in einem Array
2 $studenten = SindIn::get_studenten($_GET['grpid'], $this->env);
3 $erlaubt = count($studenten) > 1;
```

Für das ausgewählte Übungsblatt wird pro Teilnehmer eine eigene Gruppe erzeugt. In der nun erscheinenden Gruppenübersicht werden statt der gemeinsamen Gruppe die neuen Einzelgruppen aufgeführt. Diese weisen zur Identifikation im Kürzel den Begriff Einzelbewertung und den Namen des jeweiligen Gruppenteilnehmers auf. Für das nächste Übungsblatt wird automatisch eine neue Gruppe angelegt, welche wieder alle Gruppenmitglieder vereint. Diese Gruppe besitzt eine neue ID, aber das Kürzel und die Punkte für andere Übungsblätter werden von der alten Gruppe übernommen.

Abfangen fehlerhafter Benutzereingaben

- ungültige Punkte:

wenn eine negative Punktzahl oder eine Zahl, die die maximale Punktzahl für eine Aufgabe überschreitet, eingegeben wird, erscheint eine Fehlermeldung

Implementierte Controller:

Tutorerg_Controller.php:

Im Konstruktor wird die übergebene Umgebung in der Controller-Klasse auf die Klassenvariable gesetzt und von dort der Login der eingeloggt Person an die eigene Klassenvariable vergeben.

Action:

- *action_listzettel()*: holt alle Übungsblätter zu der ausgewählten Veranstaltung aus der DB
- *action_listgruppen()*: holt alle Gruppen für das ausgewählte Übungsblatt der Veranstaltung und der eingeloggt Person aus der DB
- *action_listaufg()*: testet, ob die übergebenen Werte erlaubt sind, und ruft mit diesen die Methode *listaufg2()* auf
- *action_punkte()*: speichert die eingetragenen Punkte in der DB ab

```

1 foreach($aufgaben as $aufgabe) {
2     $array['VerID'] = $_REQUEST['verid'];
3     $array['GrpID'] = $_REQUEST['grpID'];
4     $array['PrfID'] = $_REQUEST['blattnr'];
5     $array['PrfTyp'] = 'U';
6     $array['AufgNr'] = $aufgabe->AufgNr;
7     $bearbeiten = BearbeitenUeb::get_bearbeiten($array, $this->env);
8     // wenn ein Feld leer geblieben ist, wird die Aufgabe mit
9     // 0 Punkte in die DB eingetragen
10    if (is_null($bearbeiten)) {
11        $array['Punkte'] = 0;
12        $bearbeiten = new BearbeitenUeb($array, $this->env);
13    }
14    // wenn die Punktzahl negativ oder höher als die max. Punkte für
15    // diese Aufgabe ist, wird eine Fehlermeldung ausgegeben und der
16    // User gelangt erneut zu dem Eintragen-Formular
17    if ($_REQUEST[$aufgabe->AufgNr]<0 || $_REQUEST[$aufgabe->AufgNr]>$aufgabe->Punkte) {
18        $error = "Ungültige Punktzahleingabe";
19        return $this->listaufg2($_REQUEST['verid'], $_REQUEST['blattnr'],
20                                $_REQUEST['grpID'],$error);
21    }
22    // bei korrekter Punkteingabe werden diese in der DB gespeichert
23    $bearbeiten->Punkte = $_REQUEST[$aufgabe->AufgNr];
24 }

```

- `action_einzel()`: erzeugt für das aktuelle Blatt Einzelgruppen und für das folgende Blatt eine neue Gruppe, die wieder alle Teilnehmer vereint
- `action_()`: default-Action

Hilfsmethoden:

- `listaufg2()`: holt sich die einzelnen Teilnehmer der ausgewählten Gruppe und die Aufgaben für das Übungsblatt mit den maximalen Punkten aus der DB
- `gesamtpunktzahl()`: berechnet die Gesamtpunktzahl für eine Gruppe und ein Übungsblatt aus den bisher in der DB eingetragenen Punkten

Implementierte Views

- `zettel_list.tpl`: listet alle Übungsblätter zu der ausgewählten Veranstaltung aus der DB auf
- `tutorGruppen_list.tpl`: listet alle Gruppen für das ausgewählte Übungsblatt der Veranstaltung und der eingeloggtten Person aus der DB auf
- `punkte_eintragen.tpl`: listet alle Gruppenteilnehmer auf und stellt ein Formular zur Punkteeingabe bereit

[top](#)

10. Personen- und Angestelltenverwaltung

von Katja Sperber und Sabine Heider

Inhalt:

- Konzept
- Funktionen innerhalb TuKS
 - Angestellten anlegen
 - Angestellten löschen
 - Persönliche Daten bearbeiten
 - Auflistung aller Angestellten
 - Detailansicht eines Angestellten
- Implementierte Models
 - Person.php
 - Angestellter.php
 - Halten.php
- Implementierte Controller
 - Angestellter_Controller.php
- Implementierte Views

Konzept:

(vergleiche ER-Diagramm in der Datenbankmodellierung)

Die Klasse **Person** verfügt über die Attribute Login, Titel, Vorname, Nachname, Email und Passwort. Über den Login, welcher maximal 8 Zeichen lang ist, lässt sich eine Person eindeutig identifizieren (Schlüssel). Nur das Attribut Titel darf unbesetzt bleiben. Die Klasse **Person** wird vollständig unterteilt in die Klassen **Angestellter** und **Student**.

Neben den von Person stammenden Attributen verfügt ein Angestellter über eine Rolle. Diese Rolle ist festgelegt auf einen Dozenten oder Übungsleiter. Zwischen Angestellten und Veranstaltungen besteht eine N:M-Beziehung **Halten**, die die beiden Schlüssel Login und VerID beinhaltet.

Funktionen innerhalb TuKS:

- Anlegen und Löschen eines Dozenten und/oder Übungsleiter
- Verändern von persönlichen Daten bei einem Dozent oder Übungsleiter
- Auflistung aller in der DB enthaltenen Angestellten
- Detailansicht eines Angestellten
- [Übungsleiter einer ausgewählten Veranstaltung zuweisen (Funktionalität wurde nachträglich direkt unter Veranstaltung anlegen bzw. bearbeiten implementiert; dazugehörige Methoden oder Views wurden mit [] gekennzeichnet)]

Angestellter anlegen:

- nur durch einen Dozenten möglich
- Bei Auswahl der Rolle sind nur 'Dozent' und 'Übungsleiter' erlaubt
 - Umsetzung in der DB: enum
 - Umsetzung im Formular (angestellten_form.tpl): Auswahlliste

```
<tr><td><label for="Rolle">Rolle:</label></td>
2  <td><select name="Rolle">
3    <option value="Dozent">Dozent
4    <option value="Uebungsleiter">Uebungsleiter
5  </select></td>
</tr>
```

- Passwort wird vor Eintrag in die DB md5-verschlüsselt und durch eine doppelte Abfrage gesichert
- überprüfen, ob alle Attribute (mit Ausnahme von Titel) einen Inhalt zurückgeben, sonst Fehlermeldung
- überprüfen, ob der Loginname maximal nur 8 Zeichen umfasst, andernfalls Fehlermeldung
- wenn alles Ok ist, wird ein neuer Angestellter und damit auch eine neue Person erzeugt

The screenshot shows a web form titled 'Angestellten anlegen'. It contains several input fields: 'Login:', 'Rolle:' (a dropdown menu currently showing 'Dozent' with 'Uebungsleiter' as an alternative option), 'Titel:', 'Vorname:', 'Nachname:', 'Email:', 'Passwort:', and 'Passwort (wiederholen:'. At the bottom of the form are two buttons: 'Einfügen' and 'Abbrechen'.

Abb. 22: Formular zum Anlegen eines Angestellten

Angestellter löschen:

- nur Dozenten sind zum Löschen anderer Angestellten berechtigt
- als Delete-Option innerhalb der Angestelltenliste möglich
- es folgt eine Sicherheitsabfrage, ob man wirklich die ausgewählte Person löschen möchte

Persönliche Daten ändern (Dozent oder Übungsleiter)

- nur eigene Daten änderbar
- in der view *angestellter_update.tpl* werden zwei Fieldsets angezeigt, die entweder das Ändern der Daten oder des Passwortes ermöglichen
 - dadurch müssen nicht bei Änderung der Daten drei Passwortfelder ausgefüllt werden
- Ändern der Daten
 - die Datenfelder werden mit den aktuell in der DB gespeicherten Werten angezeigt
 - die Felder Login und Rolle sind mit readonly markiert, welches ein Ändern dieser unterbindet
 - bei unvollständigen Daten wird eine Fehlermeldung ausgegeben
- Ändern des Passwortes
 - die Datenfelder sind im Gegensatz zum vorherigen Fall leer
 - doppelte Abfrage des neuen Passwortes, damit nicht ausversehen ein falsches Passwort eingetragen wird
 - Bestätigung durch das alte Passwort

```

1 // Vergleich der beiden neuen Passwörter
2 if ($_POST['PasswortNeu2'] != $_POST['PasswortNeu']) {
3     $this->env->templateEngine()->assign('error', 'Passwörter stimmen nicht überein');
4     $t->assign('angestellter', $a);
5     return $t->fetch('angestellter_update.tpl');
6 }
7 // Verschlüsseln
8 $_POST['PasswortNeu'] = call_user_func(VERSCHLUESSELUNGSFUNKTION,$_POST['PasswortNeu']);

9 // wenn das alte Passwort mit dem in der DB gespeicherten
10 // übereinstimmt, wird es geändert
11 if ($a->check_passwort($_POST['PasswortAlt'])) {
12     $a->Passwort = $_POST['PasswortNeu'];
13 // sonst wird eine Fehlermeldung zurückgegeben
14 } else {
15     $this->env->templateEngine()->assign('error', 'Falsches Passwort');
16     $t->assign('angestellter', $angestellter);
17     return $t->fetch('angestellter_update.tpl');
18 }
19 $info = 'Passwort wurden erfolgreich geändert.';

```

Daten ändern

readonly Login:

Titel:

readonly Rolle:

Vorname:

Nachname:

Email:

Passwort ändern

Neues Passwort:

Neues Passwort:
(wiederholen)

Altes Passwort:

Abb. 23: Formulare zum Ändern eines Angestellten

Auflistung aller Angestellten

- Ausgabe einer Tabelle mit den Spalten Vorname, Nachname sowie den Optionen Detailansicht und Löschen
- der Login ist in der Tabelle nicht sichtbar, wird aber bei der Suche berücksichtigt

Detailansicht eines Angestellten

Details
Übersicht



Name : Willi Varchar
 Titel : dr
 Rolle : Dozent
 Email : willi@uws.de
 Login : wvarchar

Betreute Veranstaltungen

- Backen ohne Fett (SS 2009)
- Seminartest (SS 2009)

Abb. 24: Detailansicht eines Angestellten

Implementierte Models:

Person.php:

Die Klasse **Person** erweitert die abstrakte Klasse **Model** und verwaltet eine Personen-Instanz. Sie verfügt über zwei assoziative Arrays, *\$keys* und *\$properties*, in denen der Schlüssel bzw. die restlichen Attribute gespeichert werden.

Dem Konstruktor werden ein assoziatives Array mit den Attributwerten und eine Umgebung übergeben. Sofern dem Konstruktor gültige Attributwerte übergeben wurden, wird, wenn der Loginname noch nicht in der Datenbank enthalten ist, die Person in diese eingetragen, andernfalls wird ein Update durchgeführt.

statische Methoden:

- *get_person(\$login, \$env)*: liefert zu einem übergebenen Loginnamen die passende Person aus der Datenbank, falls vorhanden, andernfalls wird NULL zurückgegeben
- *get_persons(\$logins, \$env)*: liefert zu einem übergebenen numerischen Array, befüllt mit Loginnamen, ein Array mit den dazugehörigen Personenobjekten
- *get_all(\$env)*: gibt alle in der Datenbank enthaltenen Personen zurück
- *testKey(\$table, \$login, \$env)*: ermittelt zu einem übergebenen Array, das einen Loginnamen enthält, die Anzahl der Einträge in einer der Tabellen Personen, Angestellte oder Studenten. Wenn ein anderer Tabellename übergeben wird, wird eine Exception geworfen

nicht-statische Methoden:

- *insert()*: fügt die Person, falls sie noch nicht in der Datenbank enthalten ist und die Attribute mit zulässigen Werten belegt sind, dort ein
- *__get(\$property)*: gibt den Wert des übergebenen Attributes zurück
- *__set(\$property \$value)*: setzt das übergebene Attribut, sofern es sich nicht um ein Schlüsselattribut handelt, auf den übergebenen Wert und propagiert die Änderung an die Datenbank
- *update()*: führt ein Update der Datenbank durch, bei dem alle Attribute, mit Ausnahme des Schlüssels, mit den aktuell im Array gespeicherten Werten belegt werden
- *delete()*: entfernt den entsprechenden Eintrag in der Datenbank
- *check_passwort (\$password)*: übergebenes Passwort wird verschlüsselt und mit dem Passwort in der DB verglichen
- *hole_rollen()*: es werden alle Rollen der Person in einem Array zurückgegeben
- *nach_rolle(\$rolle)*: es werden die Veranstaltungen zu der entsprechenden Rollen der Person ausgegeben

Angestellter.php

Die Klasse **Angestellter** erweitert die Klasse **Person** und verwaltet eine Angestellten-Instanz. Neben den vererbten assoziativen Arrays *\$keys* und *\$properties* verfügt sie über ein weiteres assoziatives Arrays *\$angProperties*, in dem die restlichen Attribute gespeichert werden.

Dem Konstruktor werden ein assoziatives Array, das die Attribute sowohl für einen Angestellten als auch für eine Person enthält, und eine Umgebung übergeben. Sofern dem Konstruktor gültige Attributwerte übergeben wurden, wird, wenn der Loginname noch nicht in der Datenbank enthalten ist, ein Insert in Personen und Angestellte durchgeführt, andernfalls ein Update.

statische Methoden:

- *get_angestellter(\$login, \$env)*: liefert, falls vorhanden, zu einem übergebenen Loginnamen den passenden Angestellten aus der Datenbank zurück, andernfalls wird NULL zurückgegeben
- *get_angestellte(\$logins, \$env)*: liefert zu einem übergebenen numerischen Array, befüllt mit Loginnamen, ein Array mit den dazugehörigen Angestelltenobjekten zurück
- *get_all(\$env)*: gibt alle in der Datenbank enthaltenen Angestellte zurück
- *get_all_rolle(\$rolle, Umgebung \$env)*: liefert alle Angestellteinträge in der DB mit einer übergebenen Rolle

nicht-statische Methoden:

- *insert_ang()*: fügt den Angestellten, falls er noch nicht in der Datenbank enthalten ist und die Attribute mit zulässigen Werten belegt sind, in die Tabellen Personen und Angestellte ein
- *__get(\$property)*: gibt den Wert des übergebenen Attributes zurück
- *__set(\$property \$value)*: setzt das übergebene Attribut, sofern es sich nicht um ein Schlüsselattribut handelt, auf den übergebenen Wert und propagiert die Änderung an die Datenbank
- *update_ang()*: führt ein Update der Datenbank durch, bei dem alle Attribute, mit Ausnahme des Schlüssels, mit den aktuell im Array gespeicherten Werten belegt werden

Halten.php

Die Klasse **Halten** erweitert die abstrakte Klasse **Model** und verwaltet eine Halten-Instanz. Sie verfügt über ein assoziatives Arrays *\$keys*, in dem der Schlüssel gespeichert wird.

Dem Konstruktor werden ein assoziatives Array mit den Attributwerten und eine Umgebung übergeben. Wenn noch kein entsprechender Eintrag in der Datenbank vorhanden ist, wird dieser eingefügt, sofern ein existenter Loginname und eine existente VerID übergeben wurden.

statische Methoden:

- *get_halten(\$keys, \$env)*: liefert, falls vorhanden, zu einem übergebenen assoziativen Array mit den Schlüsselattributen den passenden Eintrag aus der Datenbank zurück, andernfalls wird eine Exception geworfen
- *get_veranstaltungen(\$login, \$env)*: liefert zu einem Loginname alle dazugehörigen Einträge in der Datenbank in Form eines Array mit den entsprechenden Halten-Objekten zurück
- *get_logins(\$verid, \$env)*: liefert zu einer VerID alle dazugehörigen Einträge in der Datenbank in Form eines Array mit den entsprechenden Halten-Objekten zurück
- *get_all(\$env)*: gibt alle in der Datenbank enthaltenen Einträge der Tabelle halten zurück
- *testKey(\$keys, \$env)*: gibt die Anzahl der Einträge in der Tabelle halten mit den Attributwerten des assoziativen Arrays *\$keys* halten zurück
- *testVerID(\$verid, \$env)*: gibt die Anzahl der Einträge in der Tabelle Veranstaltungen mit der übergebenen VerID zurück
- *get_ueb_ver(\$verid, \$env)*: liefert die Loginname von Übungsleitern einer übergebenen Veranstaltung in einem Array zurück

- *get_ueb_nichtver(\$verid, \$env)*: liefert die Loginname von Übungsleitern, die nicht in einer übergebenen Veranstaltung eingetragen sind, in einem Array zurück

nicht-statische Methoden:

- *insert()*: fügt die Relation, falls sie noch nicht in der Datenbank enthalten ist und die Attribute mit zulässigen Werten belegt sind, in die Tabelle halten ein
- *__get(\$property)*: gibt den Wert des übergebenen Attributes zurück
- *__set(\$property \$value)*: Diese Methode muss aufgrund der Vererbung implementiert werden, der Aufruf führt jedoch unmittelbar zu einer Exception. Die Klasse verfügt nur über Schlüsselattribute, ein Ändern dieser ist nicht gestattet.
- *delete()*: entfernt den entsprechenden Eintrag in der Datenbank

Implementierte Controller:

Angestellter_Controller.php:

Im Konstruktor wird die übergebene Umgebung in der Controller-Klasse auf die Klassenvariable gesetzt und von dort der Login der eingeloggt Person an die eigene Klassenvariable vergeben.

Action:

- *action_show()*: holt sich die Daten zu einem Login und übergibt diese an die View *angestellter_view.tpl*
- *action_new()*: zum Erzeugen eines neuen Angestellten und damit automatisch auch einer neuen Person
- *action_start()*: über eine übergebene Option wird mit "neu" das Eingabe-Formular, mit "ändern" das Update-Formular und mit "löschen" ein Fenster mit der Bestätigungsabfrage aufgerufen
- *action_delete()*: löscht einen Angestellten, wenn dies bestätigt wurde
- *action_update()*: zum Ändern der Attribute außer Passwort eines Angestellten und gegebenenfalls einer Person
- *action_passwort()*: zum Ändern des Passworts eines Angestellten und damit der Person
- *action_list()*: zur Auflistung der Angestellten
- *action_()*: Default-Action
- [*action_list_ueb()*: holt sich die Übungsleiter, welche die ausgewählten Veranstaltung halten, und jene, die die Veranstaltung nicht halten, um sie der View *uebungsleiter_list.tpl* zu übergeben]
- [*action_multizuweisen()*: trägt die durch Checkboxes markierten Übungsleiter in die Veranstaltung ein]
- [*action_multiaustragen()*: trägt die durch Checkboxes markierten Übungsleiter aus der Veranstaltung aus]

Implementierte Views:

- *angestellter_form.tpl*: Formular zur Eingabe eines Angestellten
- *angestellter_list.tpl*: Listet Angestellte auf
- *angestellter_update.tpl*: Formular zum Ändern der Angestellten Daten
- *angestellter_view.tpl*: Detailseite
- [*uebungsleiter_list.tpl*: listet die bereits der Veranstaltung zugewiesenen Übungsleiter und die verbleibenden Übungsleiter in zwei getrennten Tabellen auf]