



Lehrveranstaltungsmanagementsyste

m

Wirtschaftsinformatik, 5. Semester

Wirtschaftsinformatik-Projekt

Sommersemester 2015

Gruppenmitglieder:

Kris Klamser

Kilian Kraus

Alexander Mayer

Roland Schmid

Damian Wysocki

Inhaltsverzeichnis

1. Vorwort

2. Einleitung

2.1 Vision des Projekts

2.2 Lehrveranstaltungsmanagementsystem

3. Scrum

3.1 Rollen

3.2 Vorgehen

3.3 Artefakte

3.3.1 Product Backlog

3.3.2 Sprint Backlog

3.3.3 Burndowncharts

3.4 Projektverlauf

4. Datenbank

4.1 ER-Model

5. Lehrveranstaltungsmanagementsystem

a.

6. Abbildungsverzeichnis

7. Eigenarbeitserklärung

8. Anlagen

Ausarbeitung

1. Vorwort

Die vorliegende Dokumentation beschreibt den Verlauf und die Funktionalitäten des im Sommersemester 2015 programmierten Lehrveranstaltungsmanagementsystems.

Zunächst werden einleitend die zu Semesterbeginn formulierte Vision und die Idee hinter dem Lehrveranstaltungsmanagementsystem dargelegt. Zudem wird das Scrum-Konzept beschrieben, welches als Verfahrensrahmen für die Umsetzung des Projekts eingesetzt wurde.

Des Weiteren wird auf die Datenbank eingegangen. Hierzu wird das ER-Model und die Data Definition Language (DDL) dargestellt.

Darauffolgend wird auf die Funktionalitäten des Projekts, sowie deren Umsetzung eingegangen. Dazu werden die Funktion und der dazugehörige Quelltext beschrieben.

2. Einleitung

2.1. Vision des Projekts

Die Aufgabe besteht darin ein Lehrveranstaltungsmanagement-System für Bildungseinrichtungen zu programmieren. Hierbei sollen Veranstaltungen, die von Dozenten geleitet werden, unter verschiedenen Bedingungen den vorhandenen Räumlichkeiten zugewiesen werden. Die zur Verfügung stehenden Räumlichkeiten können in verschiedenen Gebäuden bzw. an verschiedenen Standorten liegen. Es soll jedoch vermieden werden, dass zwischen zwei aufeinanderliegenden Lehrveranstaltungen eine größere Distanz liegt. Des Weiteren sind Dozenten eventuell beruflich noch anderweitig eingebunden und können nur zu vorgegebenen Zeiten ihre Lehrveranstaltungen abhalten. Diese und noch weitere Bedingungen sind bei diesem zu programmierenden System zu berücksichtigen.

Als Ergebnis sollen den einzelnen Studenten/Schüler, sowie den Dozenten, Tutoren, Hilfskräften, wissenschaftlichen und technischen Mitarbeitern ein individueller Stundenplan sowie freie Räume ausgegeben werden.

Auch Mitarbeiter im Sekretariat sollen in ihren Aufgaben am Lehrveranstaltungsplan nachgehen und Auskünfte geben können.

2.2. Lehrveranstaltungsmanagementsystem

Unter einem Lehrveranstaltungsmanagementsystem verstehen wir ein Softwaresystem, welches Lehrveranstaltungen und deren Drumherum verwaltet. Dazu gehören nämlich nicht nur die Veranstaltungen selbst mit ihren Attributen. Auch die Fakultäten sowie die Studiengänge müssen verwaltet werden. Des Weiteren gehört auch die Bildungseinrichtung dazu. Die dazugehörigen Gebäude, Räume und Personen müssen betrachtet werden. Zu den Personen zählen dabei neben den Studenten(-innen)/Schülern(-innen) und Dozenten(-innen)/Lehrkräften auch die Mitarbeiter. Diese Rollen in diesem System sollten dabei unterschiedliche Funktionalitäten bedienen können.

Alles in allem müssen also viele Dinge beachtet und verwaltet werden. Eine der Kernaufgaben eines Lehrveranstaltungsmanagementsystems ist es zudem Stundenpläne zu erstellen und dabei zeitliche und räumliche Faktoren zu beachten.

Außerdem sollte ein solches System auch mit den Noten umgehen können, welcher jeder Student(-in)/Schüler(-in) in einer von ihm/ihr belegten Veranstaltung bekommt.

3. Scrum

Das Projekt zu diesem Lehrveranstaltungsmanagementsystem wurde im Rahmen des Scrum-Verfahrens umgesetzt. Scrum ist ein Rahmenwerk für das Projektmanagement. Im folgenden Kapitel wird das SCRUM-Konzept erklärt und die Umsetzung des Projekts beschrieben und dargestellt.

3.1. Rollen

In Scrum gibt es drei Rollen bzw. Aufgabenbereiche. Den *Product Owner*, den *Scrum Master* und das *Entwicklerteam*.

Der *Product Owner* ist der Verantwortliche auf Seite des Kunden, der die Anforderungen und das Ziel kennt. Die Anforderungen werden dabei in User Storys formuliert und zudem priorisiert. Aufgabe dieser Rolle ist es das Produkt am Ende jedes Sprints und der gesamten Entwicklung abzunehmen bzw. Abweichungen vom SOLL-Zustand anzumerken und Verbesserungen zu fordern. Diese Rolle übernahm in diesem Projekt Roland Schmid.

Der *Scrum Master* übernimmt die Leitung im Scrum-Verfahren. Er steht in engem Kontakt zum *Product Owner* und teilt die User Stories nach ihrer Priorität in einzelne Sprints ein. Zusätzlich wird der Aufwand für die Umsetzung einer solchen Anforderung bewertet. Des Weiteren ist er dafür verantwortlich, dass die Sprints umgesetzt werden und damit das Projekt nach und nach realisiert wird. Diese Rolle übernahm in diesem Projekt Kris Klamser.

Als dritte Rolle ist das *Entwicklerteam* zu nennen. Dieses besteht aus den Entwicklern, die die Funktionalitäten des Projekts interpretieren und umsetzen. Dabei werden die User Stories des jeweils aktuellen Sprints bearbeitet. Diesen Aufgabenbereich deckte in unserem Projekt die gesamte Gruppe ab.

3.2. Vorgehen

Der Ablauf bei Scrum ist prinzipiell wie in der folgenden Abbildung vorgesehen.

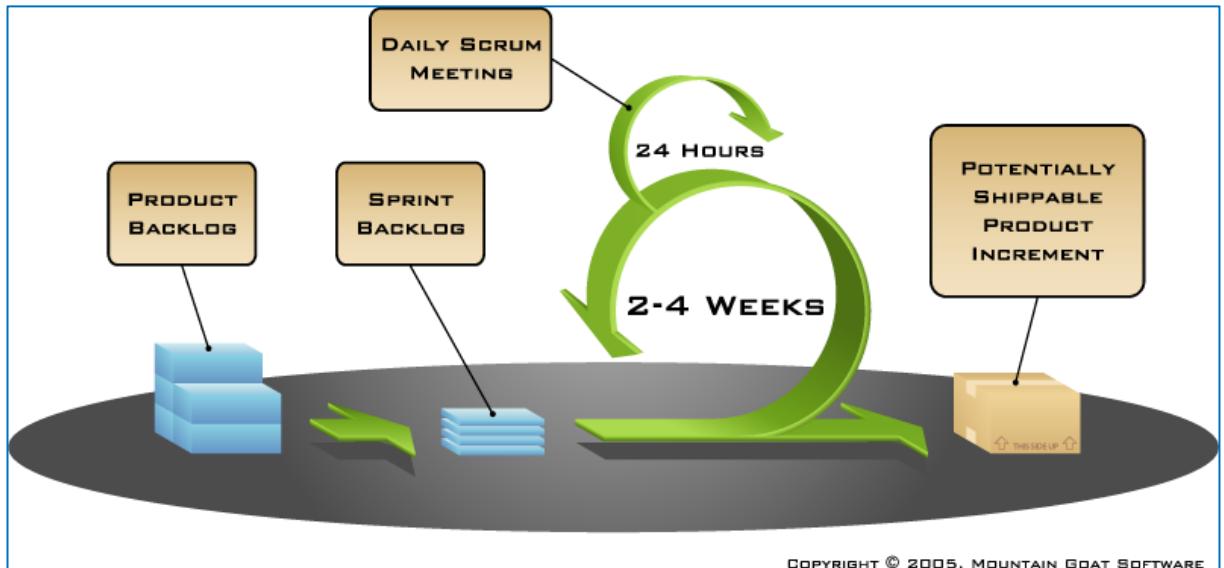


Abbildung 1: Ablauf Scrum
(<https://www.mountaingoatsoftware.com/system/asset/file/17/ScrumLargeLabelled.png>)

Der Product Backlog beinhaltet die Anforderungen des Kunden an das System in Form von User Stories und wird vom Product Owner verwaltet. Eine User Story hat dabei den grundsätzlichen Aufbau:

Als *NUTZER* will ich *FUNKTION/EIGENSCHAFT* (, damit *NUTZEN*).

Diese User Stories werden priorisiert und anhand ihrer Priorität den Sprints zugeordnet. Ein Sprint ist dabei ein Zeitraum von 2 bis 4 Wochen, in dem das Entwicklerteam versucht die Funktionalitäten, die die User Stories beschreiben, herzustellen. Ist ein Sprint zu Ende wird ein neuer Sprint mit weiteren User Stories gestartet. Dieser Vorgang wird bis Projektende und Abnahme des Produkts durch den Kunden wiederholt. Eine nicht beendete User Story kann dabei erneut in einen Sprint aufgenommen werden. Außerdem ist nicht festgelegt wie viele Sprints für ein Projekt gebraucht werden. Je nach Umfang des Projekts und Größe des Entwicklerteams kann das variieren. Verwaltet werden die Sprints vom Scrum Master.

Das Ergebnis jedes Sprints sollen voll funktionsfähige Bausteine des Systems sein, die der Product Owner abnehmen muss.

Außerdem sind regelmäßige Meetings wichtig. Die sogenannten *Daily Scrum Meetings* finden wie der Name schon sagt idealerweise täglich statt. In ihnen werden die Fortschritte ausgetauscht und eventuelle Probleme besprochen.

Abweichungen des Lehrveranstaltungsmanagement-Projekts zu diesem prinzipiellen Scrum-Ablauf werden in Punkt 3.4 „Projektverlauf“ geschildert.

3.3. Artefakte

Für das Lehrveranstaltungsmanagement-Projekt haben sich die Artefakte Product Backlog, Sprint Backlogs und Burndown Charts wie folgt ergeben.

3.3.1. Product Backlog

Im Product Backlog werden alle User Storys des Softwareprojekts verwaltet. Jede User Story bekommt dabei eine Bewertung nach Komplexität und wird priorisiert. Im Lehrveranstaltungsmanagementsystem gab es User Storys, die mehrmals in Sprint Backlogs landeten und auch ein paar, die in keinem Sprint Backlog landeten.

User Story	Bewertung	Priorität
Als Mitarbeiter möchte ich Noten von Veranstaltungen für die Teilnehmer eintragen können.	20	hoch
Als Student möchte ich mir ein Zeugnis mit allen bisher erbrachten Leistungen erstellen lassen können.	13	hoch
Als Admin/Verwalter möchte ich Rechte vergeben können.	8	hoch
Als Entwickler möchte ich eine einheitliche Optik und einen einheitlichen Aufbau der Webseiten haben.	13	hoch
Als Entwickler möchte ich ein einheitliches ER-Modell haben. (Alex)	8	hoch
Als Benutzer möchte ich die für eine Veranstaltung benötigte Ausstattung	3	hoch

festlegen können.			
Als Mitarbeiter möchte ich Räume anlegen können. (Nacharbeit)	20	hoch	
Als Entwickler möchte ich die SQL-Skripte zentral verwaltet haben.	1	hoch	
Als Mitarbeiter möchte ich einer Veranstaltung einen Raum manuell zuordnen können. (Raum und Zeit!) (Nacharbeit)	13	hoch	
Als Mitarbeiter möchte ich Lehrverantwortlicher / Dozent / Tutor zu Veranstaltung hinzufügen können. (Nacharbeit)	13	hoch	
Als Mitarbeiter möchte ich Studiengänge und Fakultäten anlegen können. (Nacharbeit)	20	hoch	
Als Mitarbeiter möchte ich einer Veranstaltung ein Fachsemester zuordnen können.	5	hoch	
Als Verwalter möchte ich Veranstaltungen anlegen können.	13	hoch	
Als Student möchte ich mich zur Verwaltung meiner Daten online einloggen können.	40	hoch	
Als Verwalter möchte ich Personen mit Rollen anlegen können.	40	hoch	
Als Dozent möchte ich mich einloggen können.	40	hoch	
Als Verwalter möchte ich Veranstaltungen ECTS vergeben können.	5	hoch	
Als Mitarbeiter möchte ich Veranstaltungen als Pflicht- und Wahlfach kategorisieren können. (Nacharbeit 2)	5	hoch	
Als Entwickler möchte ich eine einheitliche Optik und einen einheitlichen Aufbau der Webseiten haben. (Nacharbeit)	13	hoch	
Als Mitarbeiter möchte ich Räume anlegen können. (Nacharbeit 2)	20	hoch	

Als Mitarbeiter möchte ich Lehrverantwortlicher / Dozent / Tutor zu Veranstaltung hinzufügen können. (Nacharbeit 2)	13	hoch
Als Mitarbeiter möchte ich Studiengänge und Fakultäten anlegen können. (Nacharbeit 2)	20	hoch
Als Verwalter möchte ich Personen mit Rollen anlegen können. (Nacharbeit 2)	40	hoch
Als Dozent möchte ich Teilnehmerlisten erzeugen können. (Nacharbeit)	13	hoch
Als Student möchte ich mich zu Veranstaltungen an- und abmelden können.	20	hoch
Als Entwickler möchte ich einheitliche Fehlermeldungen haben.	5	hoch
Als Entwickler möchte ich im MVC-Pattern programmieren können.	40	hoch
Als Verwalter möchte ich einer Veranstaltung einem Fachsemester eines Studiengangs zuordnen können.	13	hoch
Als Entwickler möchte ich eine Verbindung von Vaadin zur Datenbank haben.	5	hoch
Als Entwickler möchte ich generische Tabellen rendern können.	8	hoch
Glassfish-Server einrichten	5	hoch
Datenbankverbindung für Vaadin erstellen	5	hoch
Als Entwickler möchte ich die Teile aus den vorigen Sprints nachbessern.	8	hoch
Als Dozent möchte ich mir mein Profil anzeigen und bearbeiten können.	13	hoch
Als Admin möchte ich alle Nutzer in einer Liste speichern können. (Nacharbeit)	13	hoch
Als Dozent/Student möchte ich mir meinen Stundenplan anzeigen können.	13	hoch
Als Dozent/Student möchte ich mir einen Stundenplan für Fachsemester anzeigen können.	13	hoch

Als Mitarbeiter möchte ich eine Veranstaltung einem Raum zuweisen unter Berücksichtigung von Überschneidungen.	8	hoch
Als Entwickler möchte ich ein Tool haben um PDFs ausgeben zu können.	8	hoch
Als Mitarbeiter möchte ich einer Veranstaltung ein Fachsemester zuordnen können. (Nacharbeit)	5	hoch
Als Verwalter möchte ich angeben können, ob für eine Veranstaltung eine Scheinleistung zu erbringen ist.	8	hoch
Als Verwalter möchte ich Veranstaltungen ändern können. (Nacharbeit)	13	hoch
Als Mitarbeiter möchte ich Noten von Veranstaltungen für die Teilnehmer eintragen können. (erneut)	20	hoch
Als Student möchte ich mir ein Zeugnis mit allen bisher erbrachten Leistungen erstellen lassen können. (erneut)	13	hoch
Als Student möchte ich online Infos zu meinem Studiengang einsehen können. (erneut)	13	hoch
Als Entwickler möchte ich die Teile aus den vorigen Sprints nachbessern. (erneut)	8	hoch
Als Entwickler möchte ich eine Datenbankstruktur haben, um den verschiedenen Rollen Eigenschaften zuweisen zu können.	20	hoch
Als Entwickler möchte ich genügend Spiel-/Testdaten zur Verfügung haben.	3	hoch
Als Mitarbeiter möchte ich einen Veranstaltungstermin löschen können.	5	hoch
Als Mitarbeiter möchte ich eine mit Fakultät Studiengang Veranstaltung codierte Veranstaltungs-ID haben.	13	hoch
Als Mitarbeiter möchte ich div. Nutzer anlegen können.	8	hoch
Als Mitarbeiter möchte ich einer	8	hoch

Veranstaltung einen Dozenten zuordnen können.			
Als Benutzer möchte ich Raumstundenpläne ausdrucken können.	8	hoch	
Als Student möchte ich meinen individuellen Stundenplan ausdrucken können.	8	hoch	
Als Mitarbeiter möchte ich einen Veranstaltungstermin ändern können.	5	hoch	
Als Benutzer möchte ich sehen, dass ich eingeloggt bin.	3	hoch	
Als Stundenplan-Planer möchte ich Lehrveranstaltungen Räume automatisch zuweisen können. (erneut)	100	hoch	
Als Entwickler möchte ich die Teile aus den vorigen Sprints nachbessern. (erneut)	8	hoch	
Als Benutzer möchte ich einen Brotkrümelpfad für die Navigation nutzen können.	8	hoch	
Als Benutzer möchte ich einen Studiengangplan anzeigen lassen können.	8	hoch	
Als Mitarbeiter möcht ich einem Studiengang Wahlfächer zuordnen können.	8	hoch	
Als Benutzer möchte ich Stundenpläne für Fachsemester ausdrucken können.	8	hoch	
Als Mitarbeiter möchte ich festlegen können, welcher Dozent welche Veranstaltungen halten kann	5	mittel	
Als Admin möchte ich alle Nutzer in einer Liste speichern können.		niedrig	
Als Mitarbeiter möchte ich Räume anlegen können.	20	niedrig	
Als Mitarbeiter möchte ich Veranstaltungen als Pflicht- und Wahlfach kategorisieren können.	5	niedrig	
Als Dozent möchte ich Teilnehmerlisten erzeugen können.	13	niedrig	

Als Dozent möchte ich E-Mails an alle Teilnehmer einer Veranstaltung verschicken können.	13	niedrig
Als Student möchte ich E-Mails an Dozenten schicken können.		niedrig
Als Verwalter möchte ich Stundenpläne automatisch erzeugen lassen können.		niedrig
Als Mitarbeiter möchte ich einer Veranstaltung einen Raum manuell zuordnen können. (Raum und Zeit!)	13	niedrig
Als Mitarbeiter möchte ich Lehrverantwortlicher / Dozent / Tutor zu Veranstaltung hinzufügen können.	13	niedrig
Als Student möchte ich mich zu Veranstaltungen an- und abmelden können.	20	niedrig
Als Mitarbeiter möchte ich Studiengänge anlegen und Veranstaltungen zuordnen können.	20	niedrig
Als Mitarbeiter möchte ich Studiengänge und Fakultäten anlegen können.	20	niedrig
Student anlegen		niedrig
Dozent anlegen		niedrig
Als Verwalter möchte ich Mitarbeiter anlegen können.	13	niedrig
Als Verwalter möchte ich Tutoren anlegen können	8	niedrig
Als Verwalter möchte ich Dozent anlegen können.	20	niedrig
Als Student möchte ich online Infos zu meinem Studiengang einsehen können.		niedrig
Als Mitarbeiter möchte ich einer Veranstaltung einen Raum unter Berücksichtigung der Ausstattung zuordnen können.	13	niedrig
Als Mitarbeiter/Dozent/Student möchte ich mir Raumpläne anzeigen lassen können.	13	niedrig
Als Entwickler möchte ich eine einheitliche	5	niedrig

Datenbankverbindung in PHP haben		
Als Mitarbeiter möchte ich einer Veranstaltung einen Raum unter Berücksichtigung der Kapazität zuordnen können.	8	niedrig
Als Benutzer möchte ich mich mit richtigen Berechtigungen einloggen können.		niedrig
Als Entwickler möchte ich Github benutzen können.		niedrig
Als Verwalter möchte ich Dozenten anlegen können		niedrig
Vereinheitlichung Tabellen E-Mail, Benutzernamen, Adressen etc.		niedrig
Als Verwalter möchte ich Veranstaltungen ändern können.	13	niedrig

3.3.2. Sprint Backlog

Die Sprint Backlogs beinhalten die User Stories, die im jeweiligen Sprint bearbeitet und umgesetzt wurden. Dies sah bei diesem Projekt wie folgt aus.

- Sprint 1:
 - Als Student möchte ich mich zur Verwaltung meiner Daten online einloggen können. (Nr. 10)
 - Als Mitarbeiter möchte ich einer Veranstaltung einen Raum manuell zuordnen können. (Raum und Zeit!) (Nr. 20)
 - Als Mitarbeiter möchte ich Lehrverantwortlicher / Dozent / Tutor zu Veranstaltung hinzufügen können. (Nr. 30)
 - Als Mitarbeiter möchte ich Studiengänge und Fakultäten anlegen können. (Nr. 50)
 - Als Verwalter möchte ich Personen mit Rollen anlegen können. (Nr. 60)
 - Student anlegen (Nr. 61)
 - Dozent anlegen (Nr. 62)
 - Als Verwalter möchte ich Mitarbeiter anlegen können. (Nr. 90)
 - Als Verwalter möchte ich Tutoren anlegen können. (Nr. 100)
 - Als Mitarbeiter möchte ich Räume anlegen können. (Nr. 110)

- Als Verwalter möchte ich Veranstaltungen anlegen können. (Nr. 120)
- Als Verwalter möchte ich Veranstaltungen ECTS vergeben können. (Nr. 121)
- Als Mitarbeiter möchte ich Veranstaltungen als Pflicht- und Wahlfach kategorisieren können. (Nr. 122)
- Als Dozent möchte ich mich einloggen können. (Nr. 140)
 - Sprint 2:
- Als Mitarbeiter möchte ich einer Veranstaltung einen Raum manuell zuordnen können. (Raum und Zeit!) (Nacharbeit) (Nr. 20a)
- Als Mitarbeiter möchte ich Lehrverantwortlicher / Dozent / Tutor zu Veranstaltung hinzufügen können. (Nacharbeit) (Nr. 30a)
- Als Mitarbeiter möchte ich Studiengänge und Fakultäten anlegen können. (Nacharbeit) (Nr. 40a)
- Als Verwalter möchte ich Personen mit Rollen anlegen können. (Nacharbeit) (Nr. 60a)
- Als Mitarbeiter möchte ich Räume anlegen können. (Nacharbeit) (Nr. 110a)
- Als Mitarbeiter möchte ich Veranstaltungen als Pflicht- und Wahlfach kategorisieren können. (Nacharbeit) (Nr. 130a)
- Als Dozent möchte ich Teilnehmerlisten erzeugen können. (Nr. 150)
- Als Mitarbeiter möchte ich einer Veranstaltung einen Raum unter Berücksichtigung der Ausstattung zuordnen können. (Nr. 160)
- Als Entwickler möchte ich eine einheitliche Datenbankverbindung in PHP haben. (Nr. 170)
- Als Verwalter möchte ich Dozenten anlegen können. (Nr. 180)
- Modifikation der Eingabemasken – Plausibilitätsprüfung. (Nr. 190)
- Vereinheitlichung Tabellen E-Mail, Benutzernamen, Adressen etc. (Nr. 200)
- Als Verwalter möchte ich Veranstaltungen ändern können. (Nr. 210)
- Als Entwickler möchte ich eine einheitliche Optik und einen einheitlichen Aufbau der Webseiten haben. (Nr. 220)
- Als Entwickler möchte ich ein einheitliches ER-Modell haben. (Nr. 230)
- Als Benutzer möchte ich die für eine Veranstaltung die benötigte Ausstattung festlegen können. (Nr. 240)
- Als Entwickler möchte ich die SQL-Skripte zentral verwaltet haben. (Nr. 250)

- Als Student möchte ich mich zu Veranstaltungen an- und abmelden können. (Nr. 260)
- Als Benutzer möchte ich mich mit richtigen Berechtigungen einloggen können. (Nr. 270)
 - Sprint 3:
- Als Mitarbeiter möchte ich Lehrverantwortlicher / Dozent / Tutor zu Veranstaltung hinzufügen können. (Nacharbeit 2) (Nr. 30b)
- Als Mitarbeiter möchte ich Studiengänge und Fakultäten anlegen können. (Nacharbeit 2) (Nr. 40b)
- Als Verwalter möchte ich Personen mit Rollen anlegen können. (Nacharbeit 2) (Nr. 60b)
- Datenbankverbindung für Vaadin erstellen. (Nr. 65)
- Als Mitarbeiter möchte ich Räume anlegen können. (Nacharbeit 2) (Nr. 110b)
- Als Mitarbeiter möchte ich Veranstaltungen als Pflicht- und Wahlfach kategorisieren können. (Nacharbeit 2) (Nr. 130b)
- Als Dozent möchte ich Teilnehmerlisten erzeugen können. (Nacharbeit) (Nr. 150a)
- Modifikation der Eingabemasken - Plausibilitätsprüfung (Nacharbeit) (Nr. 190a)
- Als Entwickler möchte ich eine einheitliche Optik und einen einheitlichen Aufbau der Webseiten haben. (Nacharbeit) (Nr. 220a)
- Als Admin möchte ich alle Nutzer in einer Liste speichern können. (Nr. 280)
- Als Mitarbeiter/Dozent/Student möchte ich mir Raumpläne anzeigen lassen können. (Nr. 290)
- Als Entwickler möchte ich Github benutzen können. (Nr. 300)
- Als Mitarbeiter möchte ich einer Veranstaltung ein Fachsemester zuordnen können. (Nr. 310)
- Als Student möchte ich mich zu Veranstaltungen an- und abmelden können. (Nr. 320)
- Als Entwickler möchte ich einheitliche Fehlermeldungen haben. (Nr. 330)
- Als Entwickler möchte ich im MVC-Pattern programmieren können. (Nr. 340)
- Als Entwickler möchte ich generische Tabellen rendern können. (Nr. 350)
- Als Verwalter möchte ich einer Veranstaltung eines Fachsemesters einen Studiengang zuordnen können. (Nr. 360)

- Glassfish-Server einrichten. (Nr. 66)
 - Sprint 4:
- Als Verwalter möchte ich Veranstaltungen ändern können. (Nacharbeit) (Nr. 210a)
- Als Admin möchte ich alle Nutzer in einer Liste speichern können. (Nacharbeit) (Nr. 280a)
- Als Mitarbeiter möchte ich einer Veranstaltung ein Fachsemester zuordnen können. (Nacharbeit) (Nr. 310a)
- Als Entwickler möchte ich die Teile aus den vorigen Sprints nachbessern. (Nr. 370)
- Als Dozent möchte ich E-Mails an alle Teilnehmer einer Veranstaltung verschicken können. (Nr. 380)
- Als Dozent möchte ich mir mein Profil anzeigen und bearbeiten können. (Nr. 390)
- Als Dozent/Student möchte ich mir meinen Stundenplan anzeigen können. (Nr. 400)
- Als Dozent/Student möchte ich mir einen Stundenplan für Fachsemester anzeigen können. (Nr. 410)
- Als Nutzer möchte ich eine Suchfunktion nutzen können. (Nr. 420)
- Als Mitarbeiter möchte ich Noten von Veranstaltungen für die Teilnehmer eintragen können. (Nr. 430)
- Als Student möchte ich mir ein Zeugnis mit allen bisher erbrachten Leistungen erstellen lassen können. (Nr. 440)
- Als Mitarbeiter möchte ich eine Veranstaltung einem Raum zuweisen unter Berücksichtigung von Überschneidungen. (Nr. 450)
- Als Entwickler möchte ich ein Tool haben um PDFs ausgeben zu können. (Nr. 460)
- Als Student möchte ich online Infos zu meinem Studiengang einsehen können. (Nr. 470)
- Als Admin/Verwalter möchte ich Rechte vergeben können. (Nr. 480)
- Als Verwalter möchte ich angeben können, ob für eine Veranstaltung eine Scheinleistung zu erbringen ist. (Nr. 490)
 - Sprint 5:
- Als Entwickler möchte ich die Teile aus den vorigen Sprints nachbessern. (erneut) (Nr. 370a)
- Als Nutzer möchte ich eine Suchfunktion nutzen können. (erneut) (Nr. 420a)

- Als Mitarbeiter möchte ich Noten von Veranstaltungen für die Teilnehmer eintragen können. (erneut) (Nr. 430a)
- Als Student möchte ich mir ein Zeugnis mit allen bisher erbrachten Leistungen erstellen lassen können. (erneut) (Nr. 440a)
- Als Student möchte ich online Infos zu meinem Studiengang einsehen können. (erneut) (Nr. 470a)
- Als Stundenplan-Planer möchte ich Lehrveranstaltungen Räume automatisch zuweisen können. (Nr. 500)
- Als Entwickler möchte ich eine Datenbankstruktur haben, um den verschiedenen Rollen Eigenschaften zuweisen zu können. (Nr. 510)
- Als Entwickler möchte ich genügend Spiel-/Testdaten zur Verfügung haben. (Nr. 520)
- Als Mitarbeiter möchte ich einen Veranstaltungstermin löschen können. (Nr. 530)
- Als Mitarbeiter möchte ich eine mit Fakultät Studiengang Veranstaltung codierte Veranstaltungs-ID haben. (Nr. 540)
 - Sprint 6:
- Als Entwickler möchte ich die Teile aus den vorigen Sprints nachbessern. (erneut) (Nr. 370b)
- Als Stundenplan-Planer möchte ich Lehrveranstaltungen Räume automatisch zuweisen können. (erneut) (Nr. 500a)
- Als Mitarbeiter möchte ich div. Nutzer anlegen können. (Nr. 550)
- Als Mitarbeiter möchte ich einer Veranstaltung einen Dozenten zuordnen können. (Nr. 560)
- Als Benutzer möchte ich Raumstundenpläne ausdrucken können. (Nr. 570)
- Als Student möchte ich meinen individuellen Stundenplan ausdrucken können. (Nr. 580)
- Als Mitarbeiter möchte ich einen Veranstaltungstermin ändern können. (Nr. 590)
- Als Benutzer möchte ich sehen, dass ich eingeloggt bin. (Nr. 600)
- Als Benutzer möchte ich einen Brotkrümelpfad für die Navigation nutzen können. (Nr. 610)
- Als Benutzer möchte ich einen Studiengangplan anzeigen lassen können. (Nr. 620)

- Als Mitarbeiter möcht ich einem Studiengang Wahlfächer zuordnen können. (Nr. 630)
- Als Benutzer möchte ich Stundenpläne für Fachsemester ausdrucken können. (Nr. 640)

3.3.3. Burndowncharts

Im Folgenden sind die Burndowncharts der einzelnen Sprints abgebildet. Diese Diagramme stellen einen SOLL-IST-Vergleich der Bearbeitung der User Stories dar. Dabei wird der geschätzte Aufwand, die sogenannten *Story Points*, als Einheit genommen und im Verhältnis zur zu Verfügung stehenden Zeit gesehen. Durch Probleme mit dem Scrum-Tool weichen die Burndowncharts eventuell etwas vom tatsächlichen Hergang der Sprints ab.

- Sprint 1:

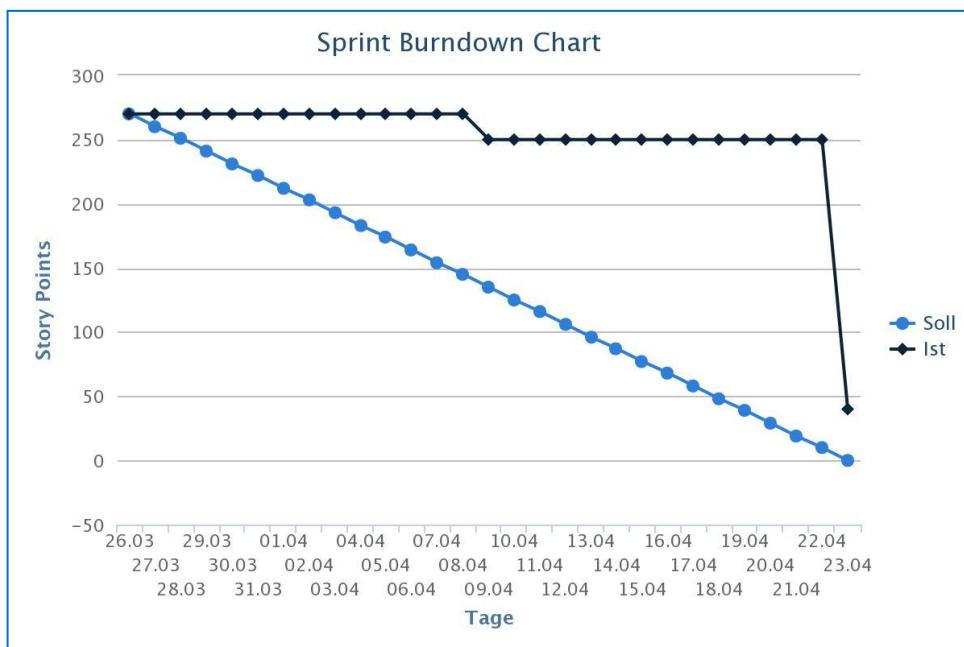


Abbildung 2: Burndown Chart Sprint 1

- Sprint 2:

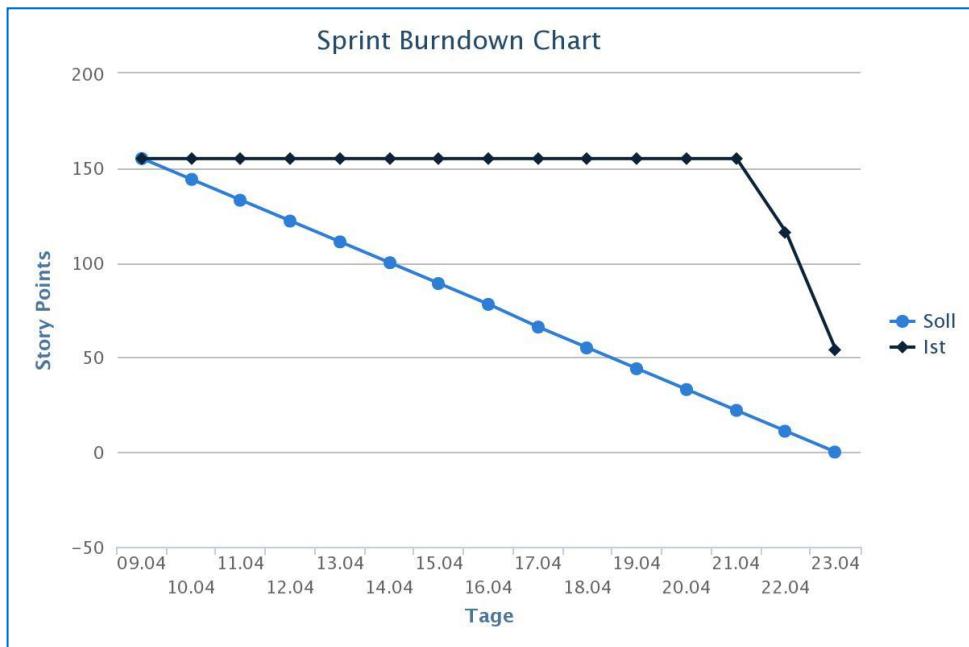


Abbildung 3: Burndown Chart Sprint 2

- Sprint 3:

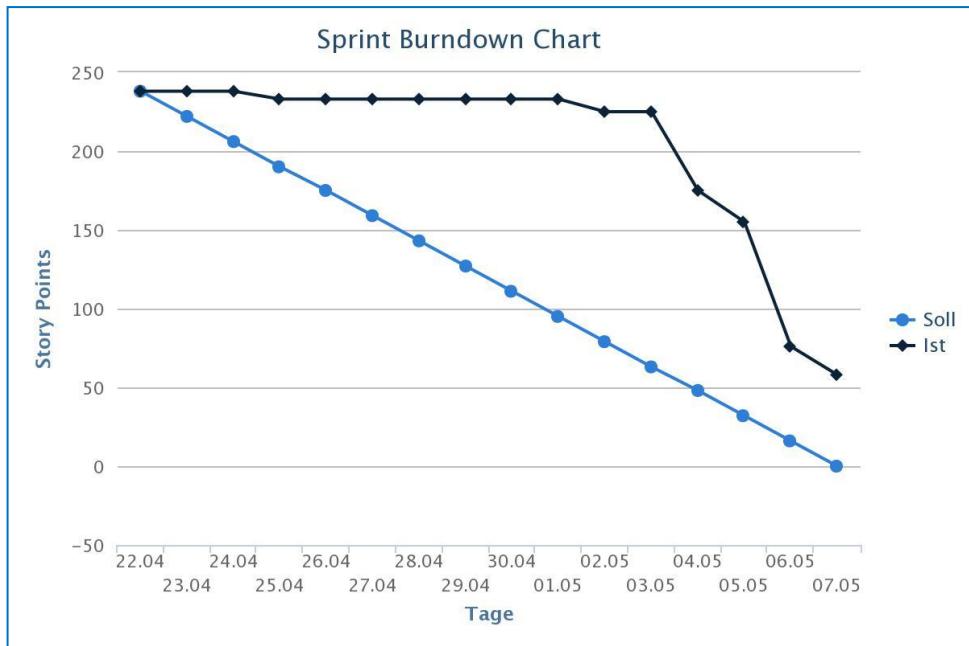


Abbildung 4: Burndown Chart Sprint 3

- Sprint 4:

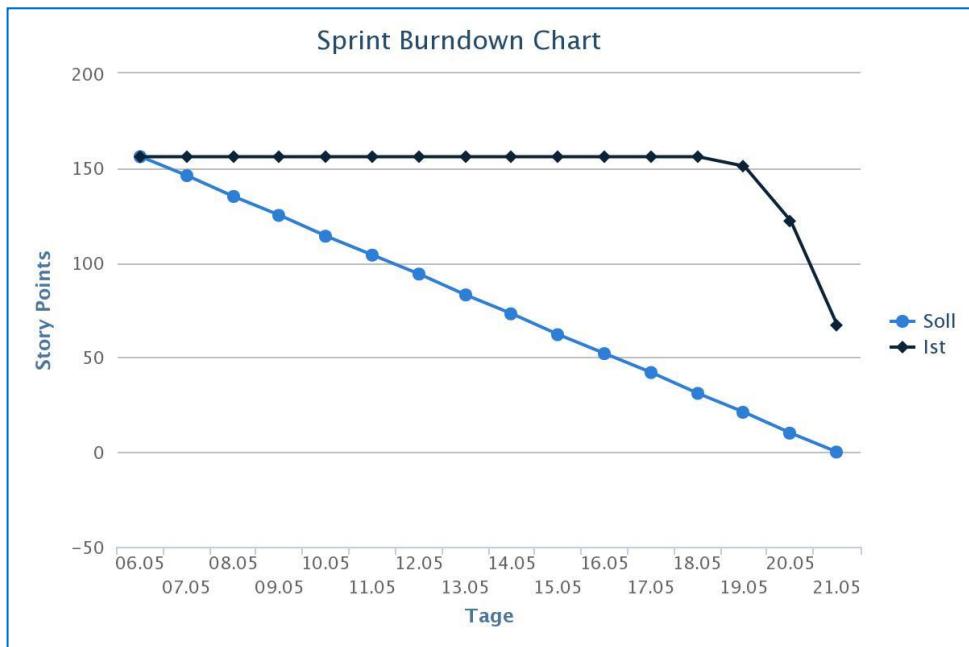


Abbildung 5: Burndown Chart Sprint 4

- Sprint 5:

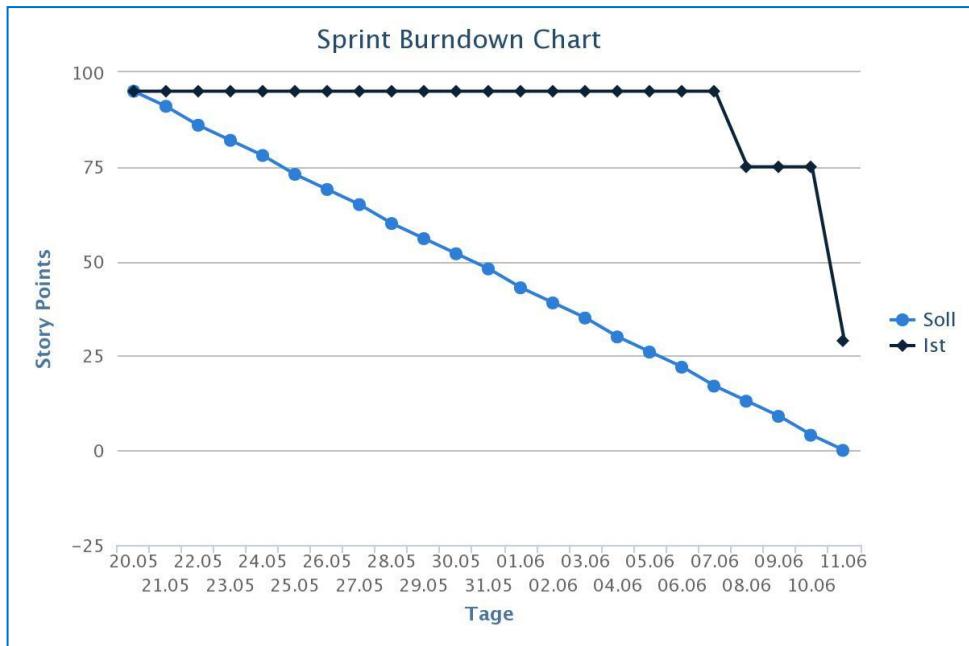


Abbildung 6: Burndown Chart Sprint 5

- Sprint 6:

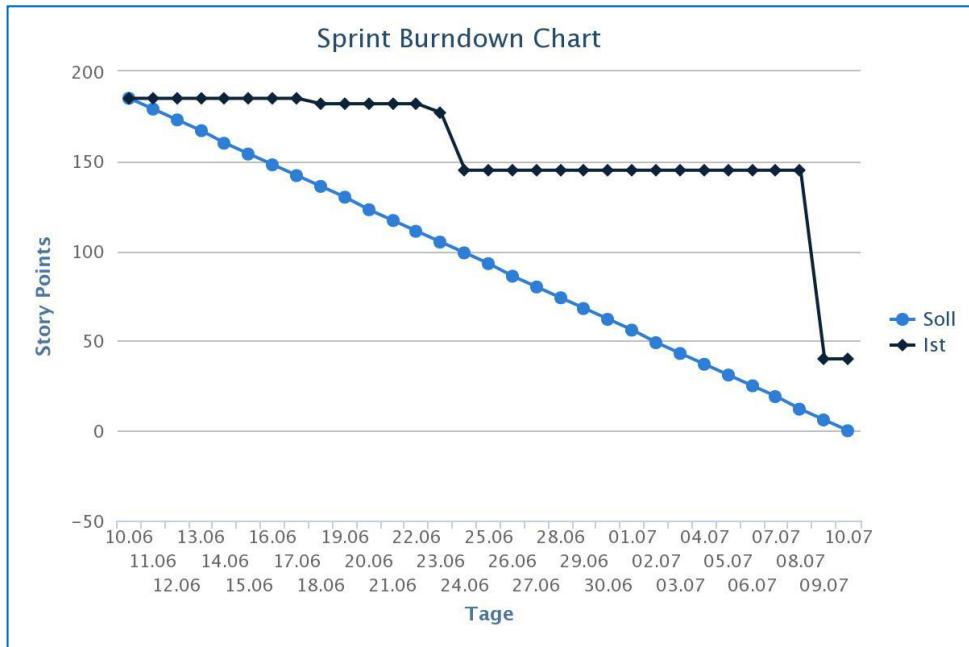


Abbildung 7: Burndown Chart Sprint 6

3.4. Projektverlauf

Das folgende Diagramm zeigt den Ablauf des Projekts über den gesamten Projektzeitraum für das Lehrveranstaltungsmanagementsystem.

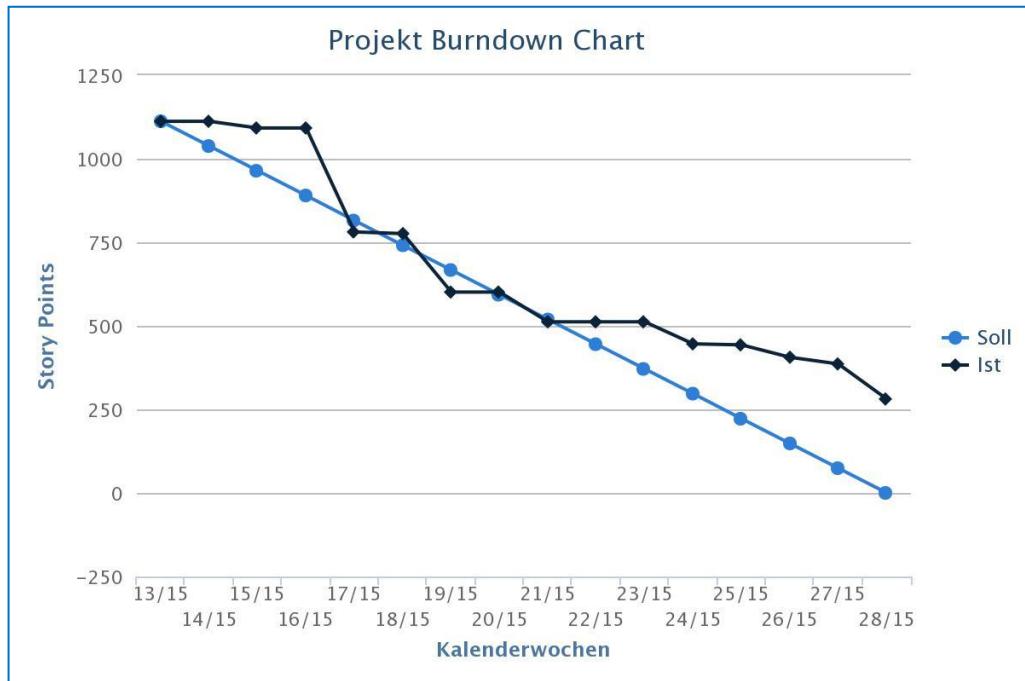


Abbildung 8: Projekt Burndown Chart

Die Sprintdauer lag dabei grundsätzlich bei zwei Wochen, lediglich der fünfte Sprint wuchs mit einer Dauer von 3 Wochen leicht ab. Während eines Sprints wurden dabei zwischen vier und sieben Scrum Meetings durchgeführt.

4. Datenbank

4.1. ER-Model

Der Datenbank-Entwurf wurde von Sprint zu Sprint kontinuierlich vorangetrieben, somit wurden mit steigender Funktionalität immer mehr Tabellen in das ER-Modell integriert, sodass erst am Ende des Projekts das finale Modell vorlag.

Als Tool zum Entwurf des Datenbankmodells wurde MySQL Workbench verwendet.

Im Folgenden wird das ER-Modell dargestellt und erläutert.

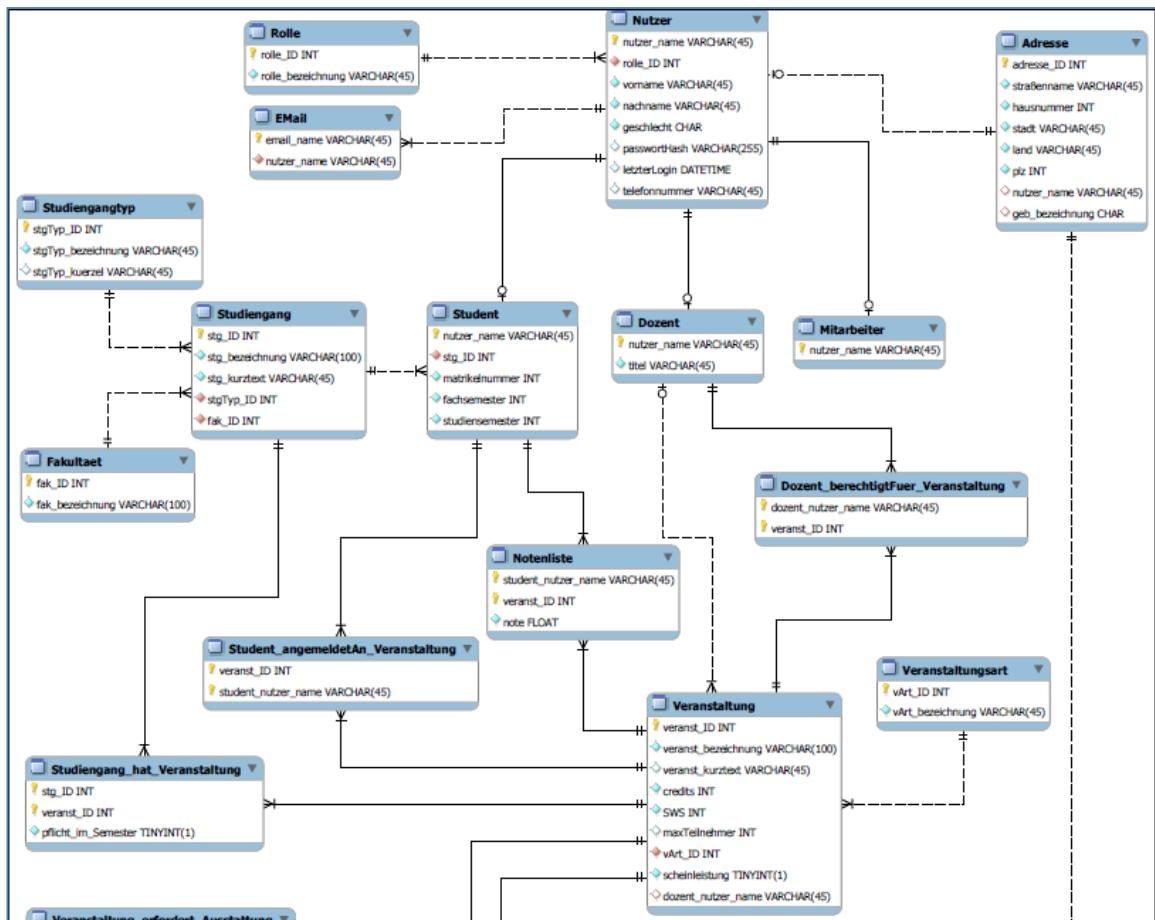


Abbildung 9: ER-Modell Teil 1

Die verschiedenen Rollen, welche ein Nutzer einnehmen kann, werden in der Tabelle Rolle abgespeichert. Jedem Nutzer ist genau eine Rolle zugeordnet.

Ein Nutzer ist entweder Student, Dozent oder Mitarbeiter. In der Nutzer-Tabelle werden Attribute wie z.B. Vorname, Nachname etc. abgespeichert, welche für jeden Nutzer von Bedeutung sind.

Ausgehend von der Nutzer-Tabelle besteht zu den Tabellen Mitarbeiter, Student und Dozent jeweils eine Spezialisierungsbeziehung (Is-a-Beziehung), sodass in diesen "speziellen" Tabellen die rollenspezifischen Eigenschaften abgespeichert werden können.

Jeder Student ist einem Studiengang zugewiesen und kann in diesem mehrere Veranstaltungen belegen. In der Tabelle Student_angemeldetAn_Veranstaltung wird abgespeichert, an welchen Veranstaltungen der Student angemeldet ist. Die Tabelle Notenliste wird benötigt, um die Noten eines Studenten bezüglich einer Veranstaltung abzuspeichern.

Ein Dozent kann im Normalfall mehrere Veranstaltungen leiten. In der Tabelle Dozent_berechtigtFuer_Veranstaltung wird abgespeichert, welcher Dozent welche Veranstaltung halten kann bzw. für welche Veranstaltung qualifiziert ist.

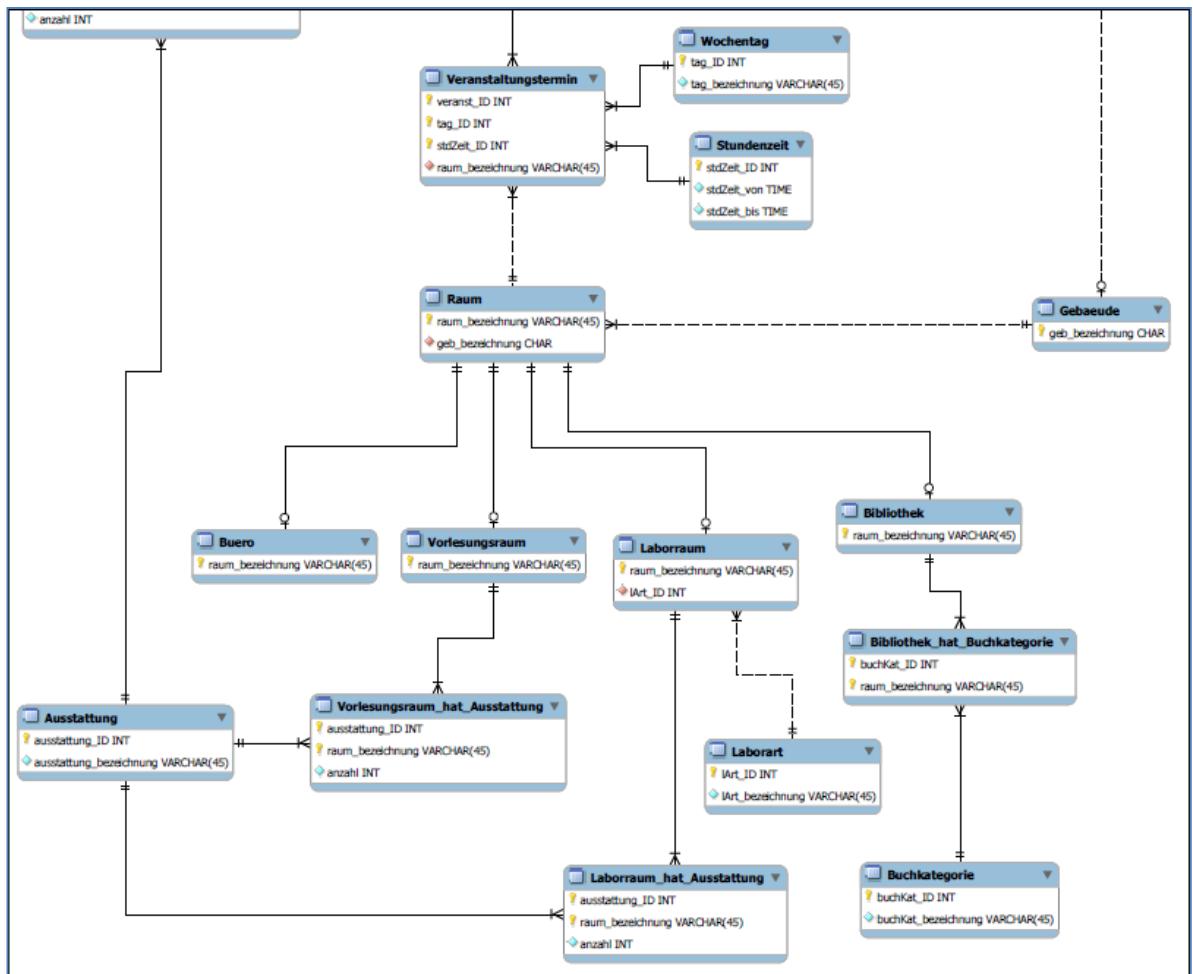


Abbildung 10: ER-Modell Teil 2

Veranstaltungstermine werden abgespeichert in der Tabelle Veranstaltungstermin. Ein Veranstaltungstermin bezieht sich auf eine Veranstaltung und findet an einem Wochentag zu einer bestimmten Stundenzzeit in einem Raum statt.

Es gibt die verschiedenen Raumarten Buero, Vorlesungsraum, Laborraum und Bibliothek.

Für Vorlesungsräume und Labore müssen abgespeichert werden, welche Ausstattungsmerkmale in welcher Anzahl im Raum vorzufinden sind, hierzu wurde die Tabelle Vorlesungsraum_hat_Ausstattung bzw. Laborraum_hat_Ausstattung angelegt.

5. Lehrveranstaltungsmanagementsystem

Im folgenden Kapitel werden die Funktionalitäten und Quelltexte des Systems erläutert. Dieses Kapitel ist enthält zunächst grundlegende Informationen und Quelltexte zum Projekt und ist dann nach den Funktionen gegliedert.

5.1 Verzeichnisstruktur

Die Anwendung besteht generell aus einem 'Public' Verzeichnis und einem 'Application' Verzeichnis. Ursprünglich war noch eine .htaccess File eingeplant, welches jedoch wegen mangelnder Kompatibilität auf dem Hochschulserver wieder rausgenommen wurde. In dem .htaccess File war ein Rewrite-Rule vorgesehen, damit die URL im Browser besser aussieht sowie eine Zugriffsbeschränkung auf das 'Application' Verzeichnis.

Das 'Application' Verzeichnis ist wiederum in 'Config', 'Controller', 'Core', 'Lib', 'Model' und 'View' aufgeteilt.

Public

Hier liegt die 'index.php' – welche als Ausgangspunkt der Anwendung fungiert – wie ebenso das CSS-File.

Application-Config

In diesem Verzeichnis befinden sich diverse Config-Files. Zum einen für die verschiedenen Laufzeitumgebungen. (Hochschulserver – public, Hochschulserver – private und Localhost). Des Weiteren befindet sich dort ein Config-File für den Emailversand.

Application-Controller

Hier liegen die Verschiedenen Controller Klassen, welche die ganze Application steuern. Diese enden alle auf '***Controller.php'

Application-Core

In diesem Verzeichnis liegen die relevanten Klassen welche für die Application relevant sind wie zum Beispiel 'Application.php', 'Environment.php', 'DatabaseFactory.php', 'Auth.php', 'View.php', 'Controller.php' usw.

Application-Lib

Im 'Lib' bzw. Library-Verzeichnis liegen externe Erweiterungen welche zum Beispiel für den E-Mail-Versand oder das erstellen eines PDF benötigt werden.

Application-Model

Hier liegen die Klassen, welche die Anbindung zur Datenbank regeln (Select, Insert, Update, usw). Diese Klassen enden alle auf '***Model.php.'

Application-View

In der View sind die Klassen welche den Output erzeugen gespeichert. In diversen Unterverzeichnissen, welche optimalerweise thematisch geordnet sind, sind die verschiedenen View Klassen abgelegt, welche auf '***View.php' enden.

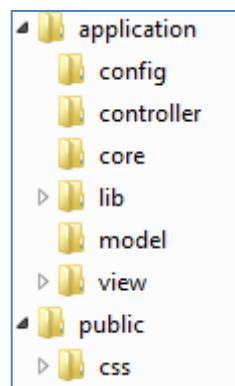


Abbildung 11: Verzeichnis

5.2 Verteilte Versionskontrolle GitHub

Bei einem Projekt mit mehreren Entwicklern empfiehlt es sich ein Datei- und Versionsverwaltungssystem bzw. Dienst zu benutzen, dass alle Veränderungen des Quellcodes dokumentiert und allen Entwicklern jederzeit zur Verfügung gestellt werden. Hierzu wurde folgende User Story formuliert:

„Nr.: xxx, Als Entwickler möchte ich GitHub benutzen können“

Für dieses Projekt hat sich eine Open-Source Variante von GitHub angeboten. Dieser Dienst bietet zum einen viele hilfreiche Funktionen und steht kostenlos zur Verfügung. GitHub ist ein webbasierter Filehosting-Dienst für Software-Entwicklungsprojekte. Namensgebend ist das Versionsverwaltungs-System Git. Im Gegensatz zu anderen Open-Source-Hostern wie SourceForge ist auf GitHub nicht

das Projekt als Sammlung von Quellcode zentral, sondern der Nutzer mit seinen Repositories (Verzeichnissen, die von Git kontrolliert werden). Um einem Repository/Projekt einen Beitrag beizusteuern, wird das Repository zunächst aufgespaltet, dann werden die zu übernehmenden Änderungen hinzugefügt und dem Besitzer des Ur-Repositorys ein Pull-Request gestellt (alles über die Weboberfläche möglich, teilweise aber auch mit Git realisierbar). Damit soll den Besonderheiten verteilter Versionskontrollsysteme besonders Rechnung getragen, und zugleich ein soziales Netzwerk geschaffen werden, was sich auch in den aus „echten“ sozialen Netzwerken bekannten Funktionen „Beobachten“ oder „Folgen“ niederschlägt [Quelle: <https://de.wikipedia.org/wiki/GitHub>]. Für dieses Projekt wurde eine eigene Organisation namens „wf5ss2015“ angelegt und ein gemeinsames Verzeichnis namens „wfprj“ erstellt. Hier wurden alle Änderungen des Quellcodes festgehalten. Dieses Verzeichnis konnte jederzeit über folgende URL eingesehen und bearbeitet werden:

<https://github.com/wf5ss2015/wfprj>

In der folgenden Abbildung ist das Startmenü von GitHub dargestellt, nachdem man sich erfolgreich eingeloggt hat. Hier kann man alle Dateien des Lehrveranstaltungssystems im Endstadium des Projektes erkennen.

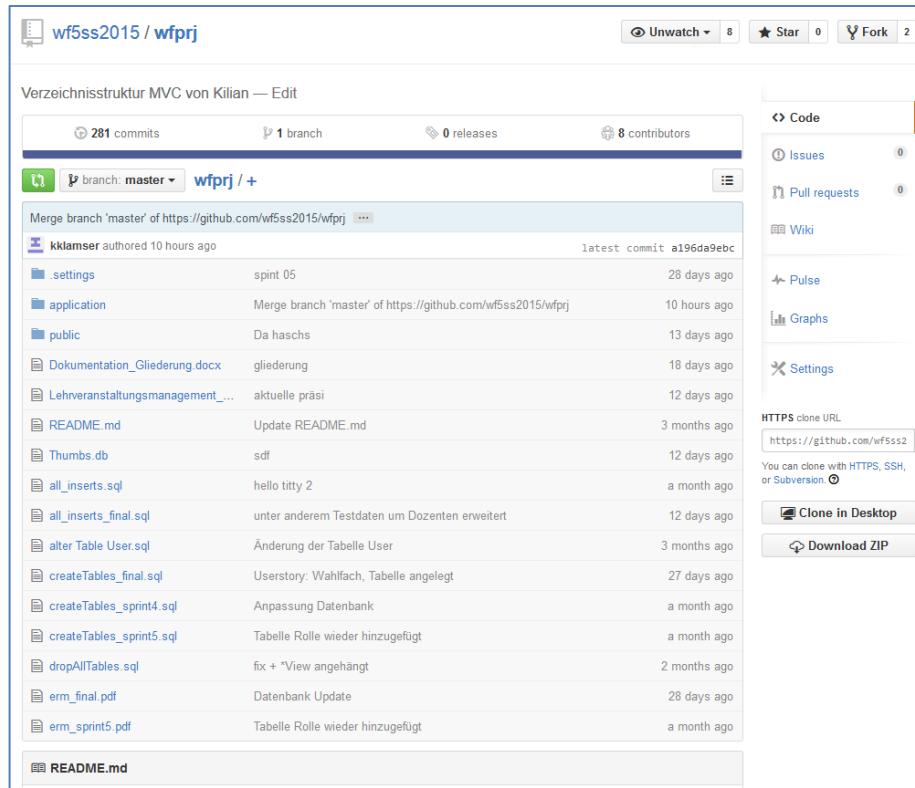


Abbildung 12: Dateiverzeichnis auf GitHub

Hilfreich war auch eine sog. „Revert“-Möglichkeit, die den geänderten Quellcode wieder rückgängig machen konnte. Auf dieser Basis wurde agile Softwareentwicklung mittels SCRUM erfolgreich für das Projekt umgesetzt.

5.3 Funktionsweise

Die ganze Application ist nach MVC (Model – View – Controller) aufgebaut und wird über 'public/index.php' gestartet welche ein neues 'Application' Objekt erstellt. Das über den Konstruktor erzeugte Application-Objekt wiederum erstellt zuerst einen Objekt eines 'Controllers' und überprüft über 'config.php' aus welcher Umgebung die Application gestartet wurde. Dies geschieht über die 'Environment.php'. Dementsprechend wird die richtige Konfiguration geladen. Der erstellte Controller wiederum lädt dann einee View und evtl. ein notwendiges Model. Beim ersten Aufruf wird ein Default-Controller wie ebenso eine Default-View geladen. Im Nachfolgenden Schaubild ist die Funktionsweise dargestellt.

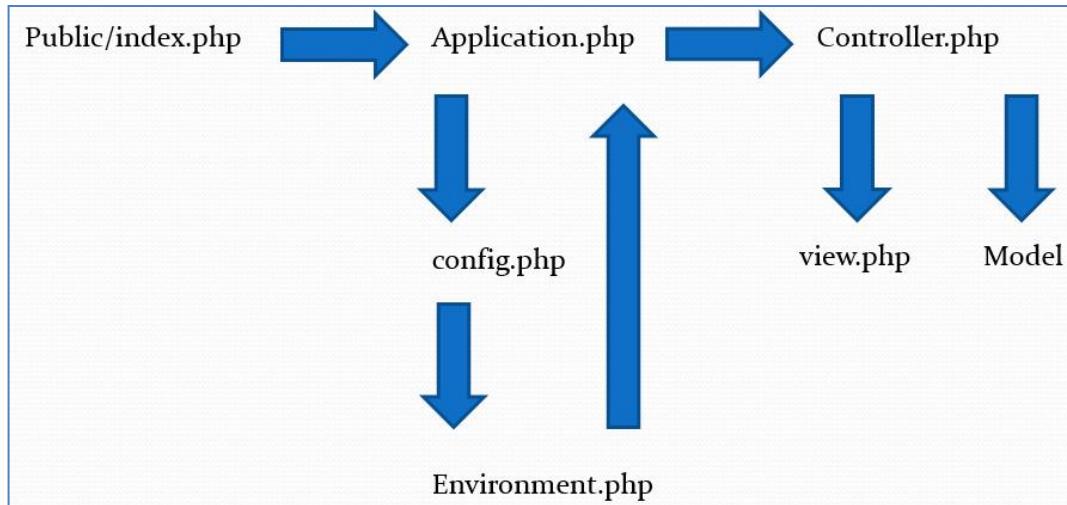


Abbildung 13: MVC

5.3.1 Applikation.php

Die 'Application.php' ist das Kernstück der Anwendung. Zuallererst werden über require die Notwendigen Klassen/Files eingebunden.

z.B.

```
require '../application/core/Config.php';
require '../application/core/Controller.php';
```

Dies ließe sich über einen Autoloader wesentlich besser handeln.

Im Konstruktor wird zum einen ein Controller-Objekt erstellt und zum anderen überprüft, ob dieser besteht. Ansonsten wird eine Fehlermeldung bzw. 404 Seite aufgerufen.

```
public function __construct() {
    $this->split ();
    $this->createController ();

    if(file_exists(Config::get ('PATH_CONTROLLER').$this->controller_name.'.php' )) {
        require Config::get ('PATH_CONTROLLER' ) . $this->controller_name . '.php';
        $this->controller = new $this->controller_name ();
    }

    if (method_exists ( $this->controller, $this->action_name )) {
        $this->controller->{$this->action_name} ();
    } else {
```

```

        header ( 'location:index.php?url=error' );
    }

} else {

header ( 'location:index.php?url=error' );
}

}

```

Die Funktion 'Split' handelt den Korrekten Umgang mit der URL. Speichert in die Klassenvariable 'controller_name' den Namen des Controllers aus der URL.

```

private function split() {

if (Request::get ( 'url' )) {

$url = trim ( Request::get ( 'url' ), '/' );

$url = filter_var ( $url, FILTER_SANITIZE_URL );

$url = explode ( '/', $url );


$this->controller_name = isset ( $url [0] ) ? $url [0] : null;

$this->action_name = isset ( $url [1] ) ? $url [1] : null;

unset ( $url [0], $url [1] );

}

}

```

Die Funktion 'creatController' erstellt einen Controller und mit der dementsprechenden 'Action' bzw. Funktion im Controller. Da die Controller-Files auf '***Controller' enden wird hier ebenso noch 'Controller' angehängt damit dies im Programm nicht extra aufgeführt werden muss.

```

private function createController() {

if ( ! $this->controller_name) {

$this->controller_name = Config::get ( 'DEFAULT_CONTROLLER' );

}

if ( ! $this->action_name or (strlen ( $this->action_name ) == 0)) {

$this->action_name = Config::get ( 'DEFAULT_ACTION' );

}

$this->controller_name = ucwords ( $this->controller_name ) . 'Controller';

}

}

```

5.3.2 Controller.php

Der Controller erstellt ein Objekt der View und initialisiert eine Session.

```
class Controller {  
    public $View;  
  
    function __construct() {  
        Session::init();  
        $this->View = new View();  
    }  
}
```

5.3.3 View.php

Die Klasse View.php erstellt ein Objekt der View welche den Output erzeugt.

Diese Funktion baut eine Seite/Output auf. Es kann ein assoziativer Array übergeben werden welcher dann in der View weiter verarbeitet werden kann. Beim Seitenaufbau ('rendern') wird zum einen der Header, das Menue, der Content an sich und ein Footer geladen.

```
public function render($filename, $data = null) {  
    if ($data) {  
        foreach ($data as $key => $value) {  
            $this->{$key} = $value;  
        }  
    }  
  
    require Config::get('PATH_VIEW') . '_templates/headerView.php';  
    new menuView();  
    $this->renderResponse();  
    require Config::get('PATH_VIEW') . $filename . 'View.php';  
    require Config::get('PATH_VIEW') . '_templates/footerView.php';  
}
```

Diese Funktion kann aus einer Instanz der View aufgerufen werden und lädt das Response Template welches Warnungen, Fehlermeldungen und Positive Rückmeldungen darstellen kann.

```
public function renderResponse() {  
    require Config::get('PATH_VIEW') . '_templates/responseView.php';
```

```
// setzt die Response auf Null, damit sie nicht zweimal angezeigt wird.

Session::set ( 'response_positive', null );
Session::set ( 'response_negative', null );
Session::set ( 'response_warning', null );

}
```

Im Prinzip die selbe Funktion wie 'render', jedoch kann dieser noch zusätzlich ein Array mit Namen mehrerer 'Views' übergeben werden damit diese doppelt zwischen Menue und Footer dargestellt werden.

```
public function renderMulti($filenames, $data = null) {

    if (! is_array ( $filenames )) {

        self::render ( $filenames, $data );
        return false;
    }

    if ($data) {

        foreach ( $data as $key => $value ) {

            $this->{ $key } = $value;
        }
    }

    require Config::get ( 'PATH_VIEW' ) . '_templates/headerView.php';
    new menueView();
    $this->renderResponse();
    foreach ( $filenames as $filename ) {

        require Config::get ( 'PATH_VIEW' ) . $filename . 'View.php';
    }
    require Config::get ( 'PATH_VIEW' ) . '_templates/footerView.php';
}
```

In nachfolgender Abbildung wird der Aufbau der View dargestellt.



Abbildung 14: Aufbau der View

5.3.4 Config.php

In dieser Klasse wird das richtige Config-File geladen. Dies geschieht über 'Environment.php'. Falls das Config-File nicht exestiert, dann gibt es einen Rückgabewert von Null bzw. false. Ansonsten wird hier die entsprechende Konfiguration geladen.

```
require 'Environment.php';

class Config {

    public static $config;

    public function get($key) {

        if (! self::$config) {

            $config_file = '../application/config/config.' . Environment::get () .
'.php';

            if (! file_exists ( $config_file )) {

                return false;

            }

            self::$config = require $config_file;
        }

        return self::$config [ $key ];
    }
}
```

5.3.5 Environment.php

Diese Klasse hat eine Funktion welche anhand des Host/URL nach vordefinierten Werten einen entsprechenden Rückgabewert liefert. In diesem Fall ist dies 'local', 'private' und 'group'. Dies wird im Falle eines Aufrufs über den Hochschulserver über den HTTP_Host und eine Zerlegung der URL erreicht.

```
class Environment {
```

```

public static function get() {

    if (getenv ( 'HTTP_HOST' ) == 'localhost') {

        return 'local';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '01') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '02') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '03') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '04') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '05') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '06') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '07') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '08') {

            return 'private';

    }

    if (getenv ( 'HTTP_HOST' ) == 'wfprj-wf5.informatik.hs-ulm.de' && substr (
$_SERVER
        ['PHP_SELF'], 1, 2 ) == '09') {

            return 'group';

    }
}

```

```

    }
}

```

5.3.6 ResponseView.php

Diese Klasse stellt den Output der Response-Message dar. Diese wird über die View (Funktion – 'renderResponse') initialisiert und bei Bedarf wird aus der Application in die Session ein String gespeichert welcher nach einmaliger Anzeige wieder gelöscht wird. Mögliche Ausgabemöglichkeiten sind positiv, negativ und warning, welche alle verschiedene Farben haben. Dies sind grün, rot und orange.

```

$response_positiv = Session::get ( 'response_positive' );
$response_negative = Session::get ( 'response_negative' );
$response_warning = Session::get ( 'response_warning' );

// output positive Rückmeldung
if (isset ( $response_positiv )) {
    foreach ( $response_positiv as $response ) {
        echo '<section class="response" id="positiv"><a style="text:bold">' .
        $response . '</a></section>';
    }
}

// output negative Rückmeldung
if (isset ( $response_negative )) {
    foreach ( $response_negative as $response ) {
        echo '<section class="response" id="negative">' . $response . '</section>';
    }
}

// output Warnung
if (isset ( $response_warning )) {
    foreach ( $response_warning as $response ) {
        echo '<section class="response" id="warning">' . $response . '</section>';
    }
}

```



Abbildung 15: Ansicht - Erfolgreich Eingeloggt

5.3.7 DatabaseFactory.php

Diese Klasse wird zum Aufbau einer Verbindung zur Datenbank verwendet. In dieser 'Factory' wird PDO zum Verbindungsauflaufbau verwendet. Hier wird im Gegensatz zur 'DatabaseFactoryMysql.php' kein Konstruktor verwendet. Der Aufbau eines Objekts der 'DatabaseFactory' erfolgt hier über die Funktion 'getFactory'. In der Funktion 'getConnection' kann über eine Instanz eine Verbindung zur Datenbank hergestellt werden.

Bei PDO gibt es diverse Konfigurationsmöglichkeiten. Wie zum Beispiel den Rückgabetyp, welcher hier auf Objekte eingestellt wurde oder den Fehlermodus. In diesem Fall wurde es so konfiguriert, dass viele Fehler angezeigt werden. In einem produktivem Einsatz sollte dies nicht der Fall sein. Dies ist eher für die Entwicklung vorgesehen.

```

class DatabaseFactory {
    private static $factory;
    private $database;
    public static function getFactory() {
        if (! self::$factory) {
            self::$factory = new DatabaseFactory ();
        }
        return self::$factory;
    }
    public function getConnection() {
        if (! $this->database) {
            $options = array (
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ,
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
            );
    }
}
  
```

```

        $this->database = new PDO ( Config::get ( 'DB_TYPE' ) . ':host=' .
Config::get ( 'DB_HOST' ) . ';dbname=' . Config::get ( 'DB_NAME' ) .
';port=' . Config::get ( 'DB_PORT' ) . ';charset=' .
Config::get ( 'DB_CHARSET' ), Config::get (
'DB_USER' ), Config::get ( 'DB_PASS' ), $options );
    }

    return $this->database;
}

}

```

5.3.8 DatabaseFactoryMysql.php

Im Gegensatz zu der anderen Klasse welche eine Datenbank Verbindung aufbaut wird hier ein Konstruktor verwendet und MySQLi verwendet. Die Verbindung über MySQLi ist im Benchmark schneller, wobei in unserer Application ist dies zu vernachlässigen. In High-Performance-Anwendungen ist dies ein Aspekt welcher zu berücksichtigen wäre. Um in der Anwendung Kompatibilität mit den Rückgabewerten von PDO herzustellen (Object-Array) wurde hier die Funktion 'query' überschrieben. Diese wandelt durch eine Schleife nun das Resultset in einen Object-Array um. Eventuell besteht dafür eine Standardfunktion, jedoch wurde hierfür keine gefunden. Ebenso wurden in dieser Klasse Fehlermeldungen implementiert, falls die Verbindung fehlschlagen sollte.

```

class DatabaseFactoryMysql extends MySQLi {

    public function __construct() {
        parent::__construct ( Config::get ( 'DB_HOST' ), Config::get ( 'DB_USER' ),
Config::get ( 'DB_PASS' ), Config::get ( 'DB_NAME' ), Config::get (
'DB_PORT' ) );

        if (mysqli_connect_error ()) {
            Redirect::to ( 'error/error' );
            Session::add ( 'response_negative', 'Error: ' . mysqli_connect_error
() );
            Session::add ( 'response_negative', 'Errno: ' . mysqli_connect_errno
() );
        }
    }

    public function query($query) {
        $result = parent::query ( $query );
        if ($this->error) {
            Redirect::to ( 'error/error' );
        }
    }
}

```

```

        Session::add ( 'response_negative', 'Error: ' . mysqli_connect_error
() );
        Session::add ( 'response_negative', 'Errno: ' . mysqli_connect_errno
() );

    }

$array = array ();
$i = 0;

while ( $obj = $result->fetch_object () ) {
    $array [$i] = $obj;
    $i++;
}

$this->close ();
return $array;
}

public function insert($insertString) {
    $success = parent::query ( $insertString );
    return $success;
}
}

```

5.3.9 Auth.php

Die Klasse Auth besteht um zu überprüfen, ob ein Anwender berechtigt ist einen Controller bzw. Funktion im Controller aufzurufen. Prinzipiell ist es möglich über die direkte URL einen Controller bzw. Funktion im Controller aufzurufen. Falls ein Anwender versucht über die URL eine Funktion im Controller aufzurufen und keine Nutzerrechte (Rolle) besitzt bzw. nicht angemeldet ist, so wird die Session zersört und der Nutzer wird auf die Login-Seite weitergeleitet. Es wäre auch möglich gewesen eine andere Aktion auszuführen.

```

class Auth {

    public function __construct($role) {
        // Initialisiert eine Session.
        Session::init ();

        if (Session::userIsLoggedIn () && Session::get ( 'user_role' ) == $role) {
            return true;
        } else {

```

```
Session::destroy ();  
  
header ( 'location: ' . "index.php?url=" . 'login' );  
  
exit ();  
  
}  
  
}  
  
}
```

Anwendungsbeispie

```
class EmployeeController extends Controller {  
    public function __construct() {  
        $auth = new Auth(3); // 3=Mitarbeiter  
        parent::__construct();  
    }  
}
```

5.3.10 Redirect.php

Die Klasse 'Redirect' ist im Prinzip ein Helper und kann zum Beispiel im Controller verwendet werden um auf die nächste Seite (Funktion im Controller) zu verweisen.

```
class Redirect {  
    public static function to($path) {  
        header ( "location: " . "index.php?url=" . $path );  
    }  
}
```

Anwendungsbeispiel

```
public function saveChangeRole() {  
    $model = new UserModel();  
  
    $model->saveRole (Request::post('rolle_ID'), Request::post('nutzer_name'));  
  
    Redirect::to ( 'employee/selectUser' );  
}  
}
```

5.3.11 Request.php

'Request' ist ebenfalls eine Helperklasse welche verwendet wird um über den Controller aus der View Werte über 'Post' geholt werden können.

```
class Request {  
  
    public static function post($key) {  
  
        if (isset ( $ POST [$key] )) {
```

```

        return $_POST [$key];
    }

}

public static function get($key) {
    if (isset ( $_GET [$key] )) {
        return $_GET [$key];
    }
}

```

Anwendungsbeispiel

```

public function saveChangeRole() {

    $model = new UserModel();

    $model->saveRole (Request::post('rolle_ID'), Request::post('nutzer_name'));

    Redirect::to ( 'employee/selectUser' );

}

```

5.3.12 Session.php

In der Session werden die Session-Daten gespeichert. Dies sind zum Beispiel der 'User_Name', 'User_Role'. Das Passwort wird z.B. nicht in der Session gespeichert, da die Session zu leicht ausgelesen werden kann und dies ein Sicherheitsrisiko wäre. Ebenso kann über die Getter und Setter aus der Application auf die Sessionvariablen zugegriffen werden bzw. auch über 'add' eine neue hinzugefügt werden. Ebenso ist es möglich die Session zu zerstören bzw. über die entsprechende Variable zu überprüfen ob der User angemeldet ist.

```

class Session {

    public static function init() {
        if (session_id () == null) {
            session_start ();
        }
    }

    public static function set($key, $value) {
        $_SESSION [$key] = $value;
    }
}

```

```

public static function get($key) {
    if (isset ( $_SESSION [$key] )) {
        return $_SESSION [$key];
    }
}

public static function add($key, $value) {
    $_SESSION [$key] [] = $value;
}

public static function destroy() {
    session_destroy ();
}

public static function userIsLoggedIn() {
    if (Session::get ( 'user_logged_in' )) {
        return true;
    } else {
        return false;
    }
}

```

5.3.13 Table.php

Diese ist eine Klasse um Tabellen generisch zu erzeugen. Als Übergabe wird mindestens ein Array mit Objekten benötigt. Optional ist es möglich zusätzlich Arrays mit einer Alias zu übergeben, welche den Spaltennamen im Output (View) der Datenbanktabelle überschreibt. Ebenso ist es möglich zusätzlich einen Array mit Links zu übergeben. Dieser Array muss dann die Action, sprich die Funktion im Controller enthalten. Wenn ein Array mit Links übergeben wurde, dann wird eine Form für jede Zeile erstellt welche die Werte aus dem 'Link'-Array enthält und diese Form enthält dann als hidden-values alle Werte der Zeile der Tabelle. Dadurch können die angezeigten Datensätze weiterverwendet werden (zum Beispiel

ändern/löschen).

```

class Table {

    public function __construct() {

    }

    public function table($data = null) {

        if ($data) {

            foreach ( $data as $key => $value ) {

                $this->{$key} = $value;

            }
        }

        echo "<table>";
        echo "<thead>";
        // falls alias übergeben wurde
        if (isset ( $this->alias )) {

            echo "<tr>";
            foreach ( $this->alias [0] as $key => $value ) {

                echo "<td>";
                echo utf8_encode ( $value );

                echo "</td>\n";
            }
            echo "</tr>\n";
            echo "</thead>";
        } else {

            // falls keine alias übergeben, dann spaltenname
            echo "<tr>";
            foreach ( $this->table [0] as $key => $value ) {

                echo "<td>";
                echo utf8_encode ( $key );
                echo "</td>\n";
            }
            echo "</tr>\n";
            echo "</thead>";
        }
    }
}

```

```

    // befüllt tabelle mit daten

    echo "<tbody>";

    echo "<tr>";

    $i = 0;

    while ( $i < count ( $this->table ) ) {

        echo "<tr>";

        foreach ( $this->table [$i] as $key => $value ) {

            if ($i % 2 != 0) {

                echo "<td style=background:#A9DFFF;>";

            } else {

                echo "<td>";

            }

            //echo utf8_encode ( $value );

            echo utf8_encode($value);

            echo "</td>\n";

        }

        // falls link

        if (isset ( $this->link )) {

            foreach ( $this->link [0] as $key => $value ) {

                $submitName = $key;

                echo "<td align=center>";

                echo "<form action="">";

                echo utf8_encode ( $value );

                echo "\" method=\"post\">";

                foreach ( $this->table [$i] as $key => $value

) {

                    echo "<input type=\"hidden\" name=\"";

                    echo utf8_encode ( $key );

                    echo "\" value=\"";

                    echo utf8_encode ( $value );

                    echo "\">";

                }

                // zusätzliche hidden values

                if(isset($this->hidden)){

                    foreach ( $this->hidden[0] as $key =>

$value ) {

```

```

echo "<input
type=\"hidden\""
name=\"";
echo utf8_encode (
$key );
echo "\ value=\"";
echo utf8_encode (
$value );
echo "\>";
}

}

echo "<input class=\"button\" type=\"submit\"
value=\"";
echo utf8_encode ( $submitName );
echo "\>";
echo "</form>";
echo "</td>\n";
}

}

echo "</tr>\n";
$i++;
}

echo "</tr>\n";
echo "</tbody>";
echo "</table>";
}

}

```

Im nachfolgenden Beispielen wird dargestellt wie sich die Tabelle aufrufen lässt und wie der Output dargestellt wird.

Aufruf ohne Alias und Link

```

$table = new Table ();
$table->table ( array (
'table' => $this->listClass,
) );

```

veranst_ID	veranst_bezeichnung	credits	SWS
1	Programmieren 1	5	4
2	Grundlagen BWL	5	4
3	Grundzüge Wirtschaftsinformatik	5	4
4	Rechnungswesen	5	4
5	Mathematik 1	5	4
6	Technologische Grundlagen	5	4
7	Programmieren 2	5	4
8	Schwerpunkt BWL	5	4
9	Business Englisch	5	4
10	Projektmanagement	5	4
11	Mathematik 2	5	4
12	Wirtschafts- und IT-Recht	5	4
13	Datenbanken	5	4
14	Business and Technical English	5	4
15	Marketing	5	4
16	ERP Systeme	5	4
17	Betriebssysteme	5	4
18	Algorithmen und Datenstrukturen	5	4

Abbildung 16: Table-Aufruf ohne Alias und Link

Aufruf mit Alias und ohne Link

```
$table = new Table ();
$alias = array (
    "0" => "ID",
    "1" => "Veranstaltung",
    "2" => "Credits",
    "3" => "SWS",
);
$alias = ( object ) $alias;

$table->table ( array (
    'table' => $this->listClass,
    'alias' => array (
        $alias
    ),
) );
```

ID	Veranstaltung	Credits	SWS
1	Programmieren 1	5	4
2	Grundlagen BWL	5	4
3	Grundzüge Wirtschaftsinformatik	5	4
4	Rechnungswesen	5	4
5	Mathematik 1	5	4
6	Technologische Grundlagen	5	4
7	Programmieren 2	5	4
8	Schwerpunkt BWL	5	4
9	Business Englisch	5	4
10	Projektmanagement	5	4
11	Mathematik 2	5	4
12	Wirtschafts- und IT-Recht	5	4
13	Datenbanken	5	4
14	Business and Technical English	5	4
15	Marketing	5	4
16	ERP Systeme	5	4

Abbildung 17: Table-Aufruf mit Alias und mit Link

Aufruf mit Alias und mit Link

```
$table = new Table ();
$alias = array (
    "0" => "ID",
    "1" => "Veranstaltung",
    "2" => "Credits",
    "3" => "SWS",
    "4" => "Anmelden"
);
$alias = ( object ) $alias;

$link = array (
    "Anmelden" => "index.php?url=student/verifyEnroll"
);
$link = ( object ) $link;
$table->table ( array (
    'table' => $this->listClass,
    'alias' => array (

```

```

    $alias
),
'link' => array (
    $link
)
);

```

ID	Veranstaltung	Credits	SWS	Anmelden
1	Programmieren 1	5	4	Anmelden
2	Grundlagen BWL	5	4	Anmelden
3	Grundzüge Wirtschaftsinformatik	5	4	Anmelden
4	Rechnungswesen	5	4	Anmelden
5	Mathematik 1	5	4	Anmelden
6	Technologische Grundlagen	5	4	Anmelden
7	Programmieren 2	5	4	Anmelden
8	Schwerpunkt BWL	5	4	Anmelden
9	Business Englisch	5	4	Anmelden
10	Projektmanagement	5	4	Anmelden
11	Mathematik 2	5	4	Anmelden
12	Wirtschafts- und IT-Recht	5	4	Anmelden
13	Datenbanken	5	4	Anmelden
14	Business and Technical English	5	4	Anmelden
15	Marketing	5	4	Anmelden

Abbildung 18: Table-Aufruf mit Alias und mit Link

5.3.14 Schedule.php

Über 'schedule.php' wird ein gültiger Stundenplan generiert. Dies umfasst soweit, dass sich keine Vorlesungen des Studiengangs, Dozent und Räume überschneiden. Als Übergabe Arrays wird ein Array mit allen Vorlesungen benötigt, welcher aus der Datenbank geladen wird. Ebenso wird ein Array benötigt in dem alle verfügbaren Räume vorhanden sind. In dem Array 'dozent' sind die Dozenten mit den dazugehörigen Vorlesungen gespeichert. (z.B. Dozent xy kann Vorlesungen x und y halten). In 'semester' wiederum sind einem Studiensemester die dazugehörigen Vorlesungen hinterlegt (z.B. WF1 hat Vorlesung x1, x2, x3, usw.).

```

class schedule{

    public $vorlesung =array ();
    public $raum=array();
    public $dozent=array();
    public $semester=array();


    function __construct($vorlesung, $raum, $dozent, $semester){

        $this->vorlesung = $vorlesung;
        $this->raum = $raum;
        $this->dozent = $dozent;
        $this->semester = $semester;

    }
}

```

Die Funktion 'match' generiert den Stundenplan. Hierzu wird zuerst eine leere Matrix mit 42 Timeslots erstellt (6 Tage * 7 Zeitfenster = 42 Slots). Über eine Schleife wird jeweils eine Vorlesungen aus dem Vorlesungen-Array geholt und versucht diese in den 'scheduleMatrix'-Array zu schreiben und dieser einen Raum und Dozenten zuzuordnen. Dies funktioniert über Zufallszahlen. Beim Zuordnen wird überprüft, ob der Slot (erste Dimension) und der Raum (zweite Dimension) frei sind. Ist dies der Fall, dann wird noch überprüft ob in diesem Slot der Dozent bzw. das Semester schon eine Vorlesung hat. Wenn ja, dann wird neu gewürfelt. Nach 100 Versuchen, welche nicht erfolgreich sind wird komplett von vorne angefangen und versucht einen gültigen Stundenplan aufzubauen.

```

function match ($obj){

    $vorlesung=$obj->vorlesung; //vorlesungen
    $raum=$obj->raum; // räume
    $dozent=$obj->dozent; // dozenten inkl. möglicher vorlesungen
    $semester=$obj->semester; // semester inkl. vorlesungen
    $scheduleMatrix[][][]=null;
    $numberOfSlots=42;
    $numberOfRooms=count($raum)-1;

    // null ist frei
    for($i=0;$i<$numberOfSlots;$i++){

```

```

$scheduleMatrix[$i][0][0]=$i;

for($k=0;$k<count($raum);$k++){
    $scheduleMatrix[$i][$k]=null;

}

$counter =0;

while (!empty($vorlesung)){
    // nach 100 versuchen von vorne beginnen
    if($counter>=100){

        $vorlesung=$obj->vorlesung; //vorlesungen
        $scheduleMatrix[][][]=null;
        // null ist frei
        for($i=0;$i<$numberOfSlots;$i++){
            $scheduleMatrix[$i][0][0]=$i;
            for($k=0;$k<count($raum);$k++){
                $scheduleMatrix[$i][$k]=null;
            }
        }
    }

    $randomR=rand(0,$numberOfRooms); //zufälliger raum
    $randomS=rand(0,$numberOfSlots-1); //zufälliger slot
    $currentV=array_pop($vorlesung); //letzte vorlesung des array
    $currentD=array(); // mögliche dozenten
    $currentS; //das semester für eine vorlesung

    $var;

    // fügt currenD array mögliche dozenten hinzu
    foreach($dozent as $key=>$value ){
        $var=$key;
        foreach($value as $key=>$value){
            if($currentV == $value){
                array_push($currentD, $var);
            }
        }
    }
}

```

```

// wählt einen möglichen dozenten aus

$selectedDozent=$currentD[rand(0,count($currentD)-1)];


// wählt das semester aus welches zur vorlesung passt

foreach($semester as $key=>$value ){

    if(in_array($currentV, $value)){

        $currentS=$key;

    }

}

//falls raum und zeit belegt sind

if(isset($scheduleMatrix[$randomS][$randomR])){

    $randomR=rand(0,$numberOfRooms); //zufälliger raum

    $randomS=rand(0,$numberOfSlots-1); //zufälliger slot

    array_push($vorlesung, $currentV);

    $counter++;

}

//falls dozent oder student keine zeit

elseif(isset($scheduleMatrix[$randomS])){

    $belegtD=0;

    $belegtS=0;

    for($i=0;$i<count($raum);$i++){

        if(isset($scheduleMatrix[$randomS][$i])){

            if($scheduleMatrix[$randomS][$i][1]===$selectedDozent){

                $belegtD=1;

            }

            if($scheduleMatrix[$randomS][$i][0]===$currentS){

                $belegtS=1;

            }

        }

    }

    if($belegtD==0&&$belegtS==0){


```

```

        $scheduleMatrix[$randomS][$randomR]=

            array(
                $currentS,
                $selectedDozent,
                $raum[$randomR],
                $currentV);
                $counter=0;

        }else{
            $randomR=rand(0,$numberOfRooms); //zufälliger
            raum

            $randomS=rand(0,$numberOfSlots-1); //zufälliger
            slot

            array_push($vorlesung, $currentV);
            $counter++;

        }

    }

    // schreibt in schedule die veranstaltungen

    else{
        $scheduleMatrix[$randomS][$randomR]=

            array(
                $currentS,
                $selectedDozent,
                $raum[$randomR],
                $currentV);

        $counter=0;
    }

}

return $scheduleMatrix;
}
}

```

Anmerkung: Dieses Verfahren ist nicht optimiert. Zum Beispiel wird nicht nach Bedingungen überprüft, welche Anforderungen eine Vorlesung an eine Raumausstattung hat. Ebenso ist es nicht nach 'Gaps' im Stundenplan optimiert (Vorlesung am Morgen – Vorlesung am Abend und nichts dazwischen). Samstag als Vorlesungstag ist als normaler Wochentag hinterlegt. Dies sollte nur der Fall sein, falls sich kein freier Slot an einem Werktag finden lassen sollte. Hierzu wurden

diverse genetische Algorithmen angeschaut welche sich hierfür besser eignen würden, wobei dies konnte leider aus Zeitgründen nicht umgesetzt werden.

5.3.15 Header.php

Im Header befindet sich soweit nur ein Login-Button welcher als Dropdown bei Auswahl ein Loginfenster öffnet. Dieser wurde über das CSS formatiert. Ebenso ist es möglich im Header noch ein Logo usw. einzubinden. Die Login-Funktionalität wurde aus folgender Quelle adaptiert und angepasst.

<http://red-team-design.com/simple-and-effective-dropdown-login-box/>

Die Funktionalität basiert auf einem einfachen Javascript welches eingebunden wurde.

Falls der Nutzer angemeldet ist, dann wird anstelle des Login-Button der Nutzernname angezeigt und die Beschriftung auf Logout abgeändert (siehe Screenshot).

```
<script src="http://code.jquery.com/jquery-latest.js"></script>
<script type="text/javascript">

$(document).ready(function(){
    $('#login-trigger').click(function(){
        $(this).next('#login-content').slideToggle();
        $(this).toggleClass('active');

        if ($(this).hasClass('active')) $(this).find('span').html('&#x25B2;');
        else $(this).find('span').html('&#x25BC;')

    })
});

</script>
</head>
<body>
<div class="wrapper"></div>
<header>
<nav id="Login">
<ul id="Login" >
<?php if(!Session::userIsLoggedIn()) {?>
```

```
<li id="Login">
  <a id="Login-trigger" href="#">
    Login <span>â–%</span>
  </a>
  <div id="login-content">
    <form action="index.php?url=Login/Login" method="post">
      <fieldset id="inputs">
        <input id="username" type="text" name="user_name" placeholder="Nutzername"
required>
        <input id="password" type="password" name="user_password" placeholder="Passwort"
required>
      </fieldset>
      <fieldset id="actions">
        <input type="submit" id="submit" value="Log in">
      </fieldset>
    </form>
  </div>
</li>

<?php }else{ ?>
<li id="Login">
  <a id="Login-trigger" href="index.php?url=Login/Logout">
    Logout (Angemeldet: <?php echo(Session::get('user_name'))?> )
  </a>
</li>
<?php }?>

</ul>
</nav>
</header>
```

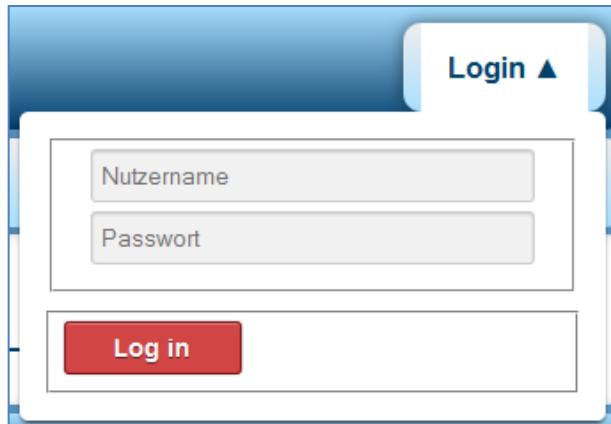


Abbildung 19: Ansicht Login-Fenster



Abbildung 20: Ansicht im eingeloggten Zustand

5.3.16 Menue.php

Die Klasse ist ein Template um das Menü aufzubauen. Dies wird in HTML5 (<nav>) erzeugt. Hierzu werden assoziative Arrays für die verschiedenen User-Rollen benötigt. Diese sind im Programmcode fest hinterlegt. Besser wäre es dies aus der Datenbank zu holen. Beispielsweise ist der Array für das Menü aufgeführt, falls der Anwender nicht angemeldet ist.

```
class menueView{  
  
    public function __construct() {  
  
        $this->menue= array(  
            // Nicht angemeldet  
            array (  
                array(  
                    "Home" => "index/index", )  
            ),  
            // Student  
            array (  
                [...]  
            ),  
    },
```

```
// Dozent
array (
    [...]
),
// Mitarbeiter
array (
    [...]
),

if (Session::get('user_role')==0){
    $this->buildMenue(0);
}else{
    $this->buildMenue(Session::get('user_role'));
}
}
```

Diese Funktion generiert das eigentliche Menü (HTML-Output). Hierbei muss noch zwischen dem ersten Menüpunkt und den Untermenüpunkten unterschieden werden, damit dies über das CSS richtig formatiert werden kann.

```
public function buildMenue($role){

$countItems = count($this->menue[$role]);
$count=0;

// baut menue
echo "<nav>";
echo "<ul>";
while($count<$countItems){

    foreach ( $this->menue [$role][$count] as $key => $value ) {
        $this->{$key} = $value;

    }
$unter =count($this->menue [$role][$count]);
$once=0;
foreach ( $this->menue [$role][$count] as $key => $value ) {

        //ohne untermenuepunkte
        if ($unter <=1){
```

```

        echo "<li>";
        echo "<a href=\"index.php?url=\"";
        echo htmlentities($value);
        echo ">";
        echo htmlentities($key);
        echo "</a>";
        echo "</li>";
    }

    // mit untermenuepunkte

    elseif($unter>1){
        // menuepunkt

        if($once==0){

            echo "<li>";
            echo "<a href=\"index.php?url=\"";
            echo htmlentities($value);
            echo ">";
            echo htmlentities($key);
            echo "</a>";
            echo "<ul>";
            $once++;
        }

        // untermenüpunkte

        elseif($once>0){

            echo "<li>";
            echo "<a href=\"index.php?url=\"";
            echo htmlentities($value);
            echo ">";
            echo htmlentities($key);
            echo "</a>";
            echo "</li>";
            $once++;
            if($once==$unter){

                echo "</ul>";
                echo "</li>";
            }
        }
    }
}

```

```

        }
    }

$count++;

}

echo "</ul>";

echo "</nav>";

}

}

```

5.3.17 ErrorController.php

Dies ist der Controller um Fehlermeldungen darzustellen. Implementiert ist soweit nur eine 404 Meldung, falls eine Aufgerufene Seite bzw. Funktion im Controller nicht besteht. Dies wird über den HTTP-header abgefragt. Das Image der Fehlermeldung war ursprünglich als Base64 codiert. Dies hat den Vorteil, dass es schneller geladen wird als ein eingebundenes Bild welches als File auf dem Host liegt.

```

class ErrorController extends Controller {

public function __construct() {

parent::__construct();

}

public function index() {

header ( 'HTTP/1.0 404 Not Found' );

$this->View->render ( 'error/index' );

}

public function error() {

header ( 'HTTP/1.0 404 Not Found' );

$this->View->render ( 'error/error' );

}

}

```



Abbildung 21: Error Controller - Seite nicht gefunden

5.3.18 IndexController.php

Der Index.Controller ist der default Controller welcher standardmäßig ausgeführt wird. Dieser beinhaltet unter anderem eine Funktion welche den Anmeldevorgang ausführt. Hierbei wird ein Objekt des 'Login-Models' erstellt welches wiederum den Eigentlichen Anmeldevorgang ausführt. Bei erfolgreicher Anmeldung wird in die Session unter der Variable 'login_successful' ein true bzw. 1 gespeichert und auf die entsprechende, rollenabhängige Seite verlinkt.. Falls der Nutzer keine gültige Rolle haben sollte, dann wird eine Fehlermeldung (Response-Message) ausgegeben. Bei der Eingabe eines falschen Passworts wird ebenfalls eine Fehlermeldung ausgegeben und wieder auf eine Login-Seite verlinkt. Es wird in dieser Fehlermeldung nicht explizit mitgeteilt, ob der Nutzer nicht existiert oder lediglich das Passwort falsch eingegeben wurde. Dies würde ein Sicherheitsrisiko darstellen, da dies für Angriffe missbraucht werden könnte.

```
public function login() {
    $model = new LoginModel();

    $login_successful = $model->login ( Request::post ( 'nutzer_name' ), Request::post (
    'nutzer_name' ) );

    if ($login_successful == 1) {

        if (Session::get ( 'user_role' ) == 3) {

            Session::add ( 'response_positive', 'Erfolgreich eingeloggt' );
        }
    }
}
```

```

Redirect::to ( 'login/helloEmployee' );

} elseif (Session::get ( 'user_role' ) == 1) {

Session::add ( 'response_positive', 'Erfolgreich eingeloggt' );

Redirect::to ( 'login/helloStudent' );

} elseif (Session::get ( 'user_role' ) == 2) {

Session::add ( 'response_positive', 'Erfolgreich eingeloggt' );

Redirect::to ( 'login/helloDocent' );

} elseif (Session::get ( 'user_role' ) == 4) {

Session::add ( 'response_positive', 'Erfolgreich eingeloggt' );

Redirect::to ( 'login/helloTutor' );

}

} else {

Redirect::to ( 'login/index' );

Session::add ( 'response_warning', 'Keine Berechtigung Sich mit diesem Account einzuloggen.' );

}

} else {

Redirect::to ( 'login/index' );

Session::add ( 'response_negative', 'Nutzer existiert nicht oder falsches Kennwort.' );

}

}

```

5.3.19 LoginModel

Das Login-Model regelt den eigentlichen Anmeldevorgang über eine Anbindung an die Datenbank. In der alten php-Version auf dem Hochschulserver war die php-Funktion 'password_verify' noch nicht implementiert. Deswegen wurde hierfür eine externe Library mit eingebunden welche diese Funktion nachbaut. In dieser wird das Passwort mit Blowfish und einem Salt gehast gespeichert. Ein solcher Hash sieht folgendermaßen aus.

\$2y\$10\$9zzYgoAvyb4KmVZNw25OXuJGWMlwxEW.7udnz6bzMKY.OM6qM8tjk2

Am Anfang von \$-Zeichen angeführt wird das Hashverfahren und die Runden dargestellt. Durch den Punkt getrennt werden der Salt und der Passwort Hash.

Nach einem erfolgreichem Anmeldevorgang werden noch die notwendigen Parameter in die Session geschrieben.

```
public function login($user_name, $user_password) {  
    // falls Nutzernname oder Passwort leer sind  
  
    if (empty ( $user_name ) or empty ( $user_password )) {  
  
        return false;  
    }  
  
    // falls Benutzer nicht in der DB besteht bzw. Passwort nicht stimmt.  
  
    $result = self::validateAnd GetUser ( $user_name, $user_password );  
  
    if (! $result) {  
  
        return false;  
    }  
  
    // Loggt Nutzer final ein  
  
    self::doLogin ( $result->nutzer_name, $result->rolle_ID );  
  
    // gibt letztendlich true zurück für erfolgreichen login  
  
    return true;  
}  
  
private function validateAnd GetUser($user_name, $user_password) {  
  
    // holt sich die Daten des Nutzers  
  
    $result = UserModel::getUserData ( $user_name );  
  
    // Überprüft ob der Nutzer besteht.  
  
    if (! $result) {  
  
        return false;  
    }  
  
    // Überprüft ob passwort mit hash Übereinstimmt.  
  
    $match = password_verify ( $user_password, $result->passwordHash );
```

```
if ($match == null) {  
  
    return false;  
  
}  
  
return $result;  
}  
  
public function logout() {  
  
Session::destroy();  
  
}  
  
public function doLogin($user_name, $user_role) {  
  
Session::init();  
  
Session::set ('user_name', $user_name );  
  
Session::set ('user_role', $user_role );  
  
Session::set ('user_logged_in', true );  
  
}  
  
public function isUserLoggedIn() {  
  
return Session::userIsLoggedIn();  
  
}
```

5.4 CSS

Im CSS ist das Design der Anwendung hinterlegt. Dies ist im Grunde nicht besonders interessant, deswegen wird nur auf einige Sachen eingegangen. Die Farbwahl hierfür wurde über einen Farbgenerator/Colorpicker im Internet getroffen, damit diese einigermaßen harmonieren. Besser wäre es gewesen das CSS für verschiedene Devices zu schreiben.

Hier wird die breite der Seite definiert sowie die Schriftgröße und der Hintergrund festgelegt. 1250 Pixel sollte mittlerweile jeder Bildschirm haben. Eine dynamische Breite wäre wie erwähnt besser.

```
body{  
    width: 1250px;  
    margin: 0 auto;  
    margin-top: 5px;  
    font-size: 100%;  
    font-family: Arial, sans-serif;  
    background: #79C5FF;  
}
```

Die Formatierung des Headers. Über 'background' wird der Hintergrund mit dem Farbverlauf festgelegt. Dies kann mittlerweile rein über CSS geregelt werden und es wird kein Image mehr benötigt. Ebenso wird aus optischen Gründen ein Schatten eingestellt.

```
/*header*/  
  
header{  
    height: 50px;  
    background: #003D6B;  
    background: Linear-gradient(top, #A9DFFF 0%, #003D6B 100%);  
    background: -moz-linear-gradient(top, #A9DFFF 0%, #003D6B 100%);  
    background: -webkit-linear-gradient(top, #A9DFFF 0%, #003D6B 100%);  
    box-shadow: 0px 0px 9px rgba(0,0,0,0.5);  
    color: #00436A;  
    border-radius: 10px;  
    text-align: center;  
    margin-top: 5px;  
    margin-bottom: 5px;  
    padding-top: 10px;  
    padding-bottom: 10px;  
}
```



Abbildung 22: CSS – Header

```
section{  
background: white;  
}  
}
```

Hier werden die Rückgabennachrichten/Response-Messages formatiert. Die ist im Grunde nur eine Formatierung des Hintergrunds.

```
background: -webkit-linear-gradient(top, #efefef 0%,#FF6600 100%); }
```

Um den Button schöner anzeigen zu können wurde dieser ebenso mit einem Farbverlauf formatiert. Des weiteren ändert er bei 'hoover' (Mauskontakt) seine Darstellung um eine Berührung für den Anwender besser ersichtlich zu machen.

```
.button {  
    cursor: pointer;  
    background: #efefef;  
    background: linear-gradient(top, #efefef 0%, #A9DFFF 100%);  
    background: -moz-linear-gradient(top, #efefef 0%, #A9DFFF 100%);  
    background: -webkit-linear-gradient(top, #efefef 0%,#A9DFFF 100%);  
    box-shadow: 0px 0px 9px rgba(0,0,0,0.50);  
    border:none;  
    padding: 2px 5px 2px 5px;  
    border-radius: 7px;  
    color: #00436A;  
    font-weight:bold;  
}
```



Abbildung 23: CSS-Button

```
.button:hover {  
    color: #A9DFFF;  
    background: #00436A;  
    background: linear-gradient(top, #efefef 0%, #00436A 20%);  
    background: -moz-linear-gradient(top, #efefef 0%, #00436A 20%);  
    background: -webkit-linear-gradient(top, #efefef 0%,#00436A 20%);  
}
```



Abbildung 24: CSS-Button (hover)

Die Formatierung für das Navigationsmenü wurde aus dem Internet adaptiert, jedoch ist hierfür die Quelle leider nicht mehr auffindbar. Optisch wurde es an das Design der Anwendung angepasst. Die Menüpunkte ändern bei Berührung ebenfalls die Darstellung.

```
/*navigation*/  
  
nav ul ul{  
  
    text-transform: capitalize;  
  
    list-style-type: none;  
  
    display: none;  
  
}  
  
ul li:hover > ul{  
  
    display: block;  
  
}  
  
nav ul {  
  
    text-transform: capitalize;  
  
    width:100%;  
  
    background: #efefef;  
  
    background: linear-gradient(top, #efefef 0%, #A9DFFF 100%);  
  
    background: -moz-linear-gradient(top, #efefef 0%, #A9DFFF 100%);  
  
    background: -webkit-linear-gradient(top, #efefef 0%, #A9DFFF 100%);  
  
    box-shadow: 0px 0px 9px rgba(0,0,0,0.50);  
  
    padding: 0px ;  
  
    margin: 0px;  
  
    border-radius: 10px;  
  
    list-style: none;  
  
    position: relative;  
  
    display: inline-table;  
  
}  
  
  
nav ul:after {
```

```
content: ""; clear: both; display: block;
}

nav ul li {
    float: left;
}

nav ul li:hover {
    background: #00436A;
    background: linear-gradient(top, #eefefef 0%, #00436A 20%);
    background: -moz-linear-gradient(top, #eefefef 0%, #00436A 20%);
    background: -webkit-linear-gradient(top, #eefefef 0%, #00436A 20%);
}

nav ul li:hover a {
    color: #A9DFFF;
}

nav ul li a {
    display: block; padding: 15px 15px;
    color: #00436A; text-decoration: none;
    font-weight: bold;
}

nav ul ul {
    width: auto;
    background: #00436A; border-radius: 0px; padding: 0;
    position: absolute; top: 100%;

}

nav ul ul li {
    float: none;
    border-top: 1px solid #A9DFFF;
    border-bottom: 1px solid #A9DFFF;
    position: relative;
```

```
}

nav ul ul li a {

padding: 15px 20px;

color: #003D6B;

}

nav ul ul li a:hover {

background: #00436A;

background: Linear-gradient(top, #efefef 0%, #00436A 20%);

background: -moz-linear-gradient(top, #efefef 0%, #00436A 20%);

background: -webkit-linear-gradient(top, #efefef 0%,#00436A 20%);

}

}
```



Abbildung 25: CSS – Navigation

5.5 Nutzer anlegen

Die Nutzer solch eines Systems müssen zunächst angelegt werden, um sich später einmal einzuloggen und damit arbeiten zu können. Diese Aufgabe fällt dem Mitarbeiter zu. Daher wurde folgende Anforderung in einer User Story gestellt:

„Nr.: xxx, Als Mitarbeiter möchte ich div. Nutzer anlegen können“

Diese User Story wurde ebenfalls gemäß des MVC Patterns entwickelt. Zunächst soll die fertige Funktion im nachfolgenden Bild dargestellt werden.

Ein Nutzerkonto für "Studenten" anlegen

Bitte alle folgenden Felder ausfüllen!

Matrikel	9111114
(wird automatisch vergeben!)	
Vorname	
Nachname	
Geschlecht	<input type="radio"/> w <input checked="" type="radio"/> m
Telefonnummer	
Straße	
Hausnummer	
Land	Deutschland
Stadt	
PLZ	
Email	
Nutzername	
(Falls Nutzer bereits vorhanden, wird eine fortlaufende numerische Ziffer angehängt!)	
Passwort	
Passwort wdh	
Studiengang	Digital Media
Studiensemester	
Fachsemester	

Studenten anlegen

copyright

Abbildung 26: View Studenten anlegen

In **Abbildung 10** wird der Quellcode der Datei „**AnlegenStudentView.php**“ aus dem Ordnerpfad „application/view/employee“ angezeigt. Sobald dieser Menüpunkt des Lehrveranstaltungsmanagementsystems aufgerufen wird, werden bereits die letzte gültige Matrikelnummer und die hinterlegten Studiengänge aus der Datenbank extrahiert und im dem Formular angezeigt. Dies wird wieder über den Controller gesteuert, der sich in der Datei „EmployeeController.php“ und dem Ordnerpfad „application/controller“ befindet. In **Abbildung 11** wird die dazugehörige Funktion „zeigeStudentAnlage“ angezeigt.

```
public function zeigeStudentAnlage() {
    $model = new UserModel();
    $model1 = new UserModel();
    $this->View->render ( 'employee/AnlegenStudent', array (
        'matrikel' => $model->getMatrikel (), 'studiengaenge' => $model->getStudiengaenge ()
    ) );
}
```

Abbildung 27: Controller Student anlegen

Hier werden zunächst zwei Objekte der UserModel-Klasse angelegt aus der Datei „**UserModel.php**“. Als nächstes werden zwei Arrays „**matrikel**“ und „**studiengaenge**“ jeweils für die letzte gültige Matrikelnummer und alle hinterlegten Studiengänge erstellt. Dort werden dann über den Aufruf der Funktionen aus dem Model „**getMatrikel**“ und „**getStudiengaenge**“ die entsprechenden Daten gespeichert und an die View übergeben. Die zwei Methoden aus dem Model werden in **Abbildung 12** und **Abbildung 13** dargestellt.

```
public function getMatrikel() {
    $database = DatabaseFactory::getFactory ()->getConnection ();

    $sql = "SELECT MAX(matrikelnummer) as matrikelnummer FROM student
            GROUP BY matrikelnummer";
    $query = $database->prepare ( $sql );
    try{
        $query->execute ();

        return $query->fetch();
    } catch(PDOException $e){
        Session::add ( 'response_negative', 'Matrikelnummer kann nicht ausgelesen werden.', $e );
    }
}
```

Abbildung 28: Letzte aktuelle Matrikelnummer extrahieren

Die Funktion „**getMatrikel**“ beinhaltet ein SELECT Statement mit einer Aggregatfunktion, welche die größte Matrikelnummer aus der Datenbank ausliest. Dieser Wert wird dann in das entsprechende Feld der View bei Aufruf eingefügt. In der Folgenden Abbildung ist die Funktion „**getStudiengaenge**“ beschrieben. Hier werden anhand der Studiengangs-ID, Studiengangs-Bezeichnung aus der Tabelle Studiengang, alle verfügbaren Studiengänge als Dropdown Liste in dem entsprechenden Feld der View eingefügt. Auch hier wird ein einheitliches Fehlerhandling verwendet.

```
public function getStudiengaenge() {
    $database = DatabaseFactory::getFactory ()->getConnection ();

    $sql = "SELECT stg_ID, stg_bezeichnung FROM studiengang";
    $query = $database->prepare ( $sql );
    try{
        $query->execute ();
        return $query->fetchAll ();
    } catch(PDOException $e){
        Session::add ( 'response_negative', 'Studiengänge können nicht angezeigt werden.', $e );
    }
}
```

Abbildung 29: Alle hinterlegten Studiengänge auslesen

Nachdem in der View die beiden Felder Matrikelnummer und Studiengänge als Vorauswahl angezeigt werden, kann nun ein Student angelegt werden. In **Abbildung 12** wird ein Auszug des entsprechenden Quellcodes der Datei „**AnlegenStudentView**“ angezeigt.

```

<article>
    <h1>Ein Nutzerkonto für <b>"Studenten"</b> anlegen</h1>
    <br>
    <br>
    <a>Bitte <b>alle</b> folgenden Felder ausfüllen! </a>
    <br>
    <br>

    <?php/* Javascript Funktionen zum generieren eines Nutzernamens und
        Überprüfen der Password Felder auf Gleichheit*/?>
    <script>
        function generateUserName() {
            if(document.form.username.value=='' && document.form.fname.value!=''
                && document.form.lname.value!='') {
                var username = document.form.fname.value.substr(0,2) +
                    document.form.lname.value.substr(0,3);
                username = username.replace(/\s/g, '');
                username = username.replace(/\'+/g, '');
                username = username.replace(/-/g, '');
                username = username.toLowerCase();
                document.form.username.value = username;
            }
        }

        function checkPW() {
            if(document.form.pw2.value != document.form.pw1.value){
                alert("Passwörter stimmen nicht überein");
                document.form.pw2.value = "";
                document.form.pw1.value = "";
                document.form.pw1.focus();
            }
        }

        function checkMe() {
            if (confirm("Student jetzt anlegen?")) {
                return true;
            } else {
                return false;
            }
        }
    </script>

    <?php// Formular zum Anlegen?>
<?php if ($this->matrikel and $this->studiengaenge)
{ ?>
    <form action="index.php?url=Employee/anlageStudent" name="form" method="post">
        <table>
            <?php//Matrikel?>
            <tr>
                <td><label>Matrikel</label></td>
                <td><input name="matrikel" required="required" type="text" readonly value="<?php
                    echo htmlentities($this->matrikel->matrikelnummer +1)?>"></td>
            </tr>

```

Abbildung 30: View Anlegen eines Studenten

Zunächst werden die beiden Arrays „**matrikel**“ und „**studiengaenge**“ geprüft ob sie leer sind. Ist dies nicht der Fall, wird der Wert der Matrikelnummer aus der Datenbank um eins erhöht und in der View angezeigt. In der folgenden Abbildung wird der Quellcode für das Feld Studiengang angezeigt.

```

<?php //Studiengang?>
<tr>
    <td><label>Studiengang</label></td>
    <td><select name="studiengaenge" style="max-width:34%">
        <?php foreach($this->studiengaenge as $key => $value) { ?> required="required">
            <option value="<?php echo htmlentities($value->stg_ID); ?>">
                <?php echo htmlentities($value->stg_bezeichnung); ?></option>
            <?php } ?>
        </select></td>
    </tr>

```

Abbildung 31: View Studiengänge befüllen

Hier werden die einzelnen Einträge aus dem Array „**studiengaenge**“ in das Feld des Formulars mittels einer foreach-Schleife eingefügt. Weiterhin kommen in diesem Formular drei javascript Funktionen, „**generateUserName**“, „**checkPW**“ und „**checkME**“ zum Einsatz. Die Funktion „**generateUserName**“ generiert automatisch einen Nutzernamen, sobald ein Vor- und Nachname eingegeben wurden. In diesem Fall besteht der Nutzernamen aus den ersten drei Buchstaben des Vornamens und dem Nachnamen selbst. Die zweite Funktion „**checkPW**“ prüft, ob das eingegebene Passwort mit der Wiederholung des Passworts übereinstimmt. Die letzte Funktion „**checkME**“ wird aufgerufen, sobald das Formular über den Button „**Student anlegen**“ an den Controller weitergeleitet wird. Hier soll nochmal eine Bestätigung des Nutzers über die tatsächliche Anlage des gerade eingegebenen Studenten erfolgen. Dies stellt eine weit verbreitete Handhabung solcher Formulare dar. Wenn alle Daten eingegeben sind, wird zunächst der Controller aufgerufen. Hierzu steht die Funktion „**anlageStudent**“ in der Datei „**EmployeeController.php**“ im Ordnerpfad „**application/controller**“ bereit. Hier werden die einzelnen Daten, die in dem vorherigen Formular eingetragen wurden, verarbeitet. Die einzelnen Felder sind weiterhin als „**required**“ deklariert. Somit müssen die Felder für eine erfolgreiche Anlage befüllt werden. Die beiden Typen der Felder für Fachsemester und Studiensemester sind entsprechend dem neuen HTML5 Standards als „**number**“ deklariert. In **Abbildung 16** sieht man die bereits erwähnte Funktion „**anlageStudent**“. Hier findet folgende Programmlogik statt:

- **Sicherstellen**, dass ein einzigartiger Nutzernamen gespeichert wird
- **Hashen** des dazugehörigen Passworts
- Datengruppen anlegen für:

- **Nutzerdaten:** Nutzernname, Vorname, Nachname, Telefonnummer, Geschlecht, Rolle
- **Adresse:** Nutzernname, Straße, Hausnummer, Land, PLZ
- **Email:** Nutzernname, Email
- **Studentendaten:** Nutzernname, Studiengang, Matrikel, Fachsemester, Studiensemester

```

public function anlageStudent() {

    $model = new UserModel();
    $model1 = new UserModel();
    $model2 = new UserModel();
    $model3 = new UserModel();

    // Richtigen User aus DatenBank extrahieren
    $modelUser = new UserModel();
    $strUser = $modelUser->getRightUsername(Request::post('nutzername'));

    // Passwort hashen
    $options = array();
    $hashedPassword = password_hash(Request::post('passwort'),1, $options);

    // Nutzerkonto
    $model->saveUsername(
        $strUser, Request::post('vorname'),
        Request::post('nachname'), $hashedPassword,
        Request::post('telefonnummer'), Request::post('geschlecht'),
        Request::post('rolle'));

    // Adressdaten
    $model1->saveAddress(
        $strUser, Request::post('strasse'),
        Request::post('hausnummer'), Request::post('stadt'),
        Request::post('land'), Request::post('plz'));

    $model2->saveEmail(Request::post('nutzername'), Request::post('email'));

    // Studentendaten
    $model3->saveStudentData(
        $strUser, Request::post('studiengang'),
        Request::post('matrikel'), Request::post('fachsemester'),
        Request::post('studiensemester'));
    // View erneut rendern
    Redirect::to ('employee/zeigeStudentAnlage' );
}

```

Abbildung 32: Controller Student anlegen

Die o. g. Programmlogik wird durch die einzelnen Funktionen in der **Abbildung 16** gezeigten Grafik unterstützt. Das heißt die einzelnen Abfragen an die Datenbank

werden hier gesteuert. Als erstes werden neue Objekte der UserModel-Klasse erzeugt. Danach wird der automatisch generierte Nutzernname mit der Datenbank geprüft. Sollte hier bereits ein identischer Nutzernname existieren, wird eine zufällige zweistellige Nummer angehängt. Hierfür ist die Funktion rand() von PHP innerhalb der Methode „**getRightUsername**“ zuständig. Als Parameter wird der generierte Nutzername übergeben. Im Folgenden Bild sieht man die Datenbankabfrage, die Teil der Model-Komponente darstellt.

```
public function getRightUsername($nutzer) {
    ...
    $database = DatabaseFactory::getFactory ()->getConnection ();
    ...
    // wenn Nutzer bereits vorhanden dann fortlaufende Nummer anhängen
    $sql = "SELECT max( nutzer_name )
            FROM nutzer
            WHERE nutzer_name LIKE '".$nutzer."%'";
    ...
    $query= $database->prepare ($sql);
    ...
    try {
        $query->execute ();
        ...
        $result = $query->fetchColumn ();
        ...
        if($result != $nutzer){
            $result2 = $result.rand(1,10);
            ...
            $nutzer = $result2;
        }
        ...
        //print_r($nutzer);
        return $nutzer;
    } catch ( PDOException $e ) {
        Session::add ( 'response_negative', 'Fehler: neuer Nutzernname' );
    }
}
```

Abbildung 33: Model Nutzernamen bestimmen

Zunächst wird in dem SQL-Statement die Aggregatfunktion „**MAX**“ von MySQL verwendet. Diese soll den letzten existierenden Nutzer zurückgeben, falls der generierte Nutzername bereits existiert. Die Rückgabe erfolgt mit „**fetchColumn**“, da hier nur ein Wert zurückgegeben wird. In der darauffolgenden If-Abfrage wird das Ergebnis aus der Datenbank mit dem neu generierten Nutzernamen verglichen.

Sollte es hier eine Übereinstimmung geben, wird der generierte Nutzernname um eine zweistellige Ziffer erweitert und zurückgegeben. Falls dies nicht der Fall sein sollte, wird einfach der generierte Nutzername angelegt. Die nächste Funktion, die in diesem Controller aufgerufen wird, ist die Funktion „**password_hash**“. Hier wird das eingegebene Passwort mit dem Algorithmus „**Blowfish**“ hashcodiert. Dies geschieht in der Datei „**passwordLib.php**“, welche sich im Verzeichnis „**application/lib/phppasswordlib**“ befindet. Die Funktion „**saveUsername**“ wird als nächstes im Model aufgerufen. In der folgenden Abbildung ist diese Funktion abgebildet.

```
public function saveUsername($nutzer, $vorname, $nachname, $passwort,
                            $telefonnummer, $geschlecht, $rolle) {
    $database = DatabaseFactory::getFactory ()->getConnection ();

    $sql = "INSERT INTO nutzer(nutzer_name, vorname, nachname, rolle_ID,
                           passwortHash, telefonnummer, geschlecht) VALUES('".$nutzer."',
                           '".$vorname."', '".$nachname."', '".$rolle."',
                           '".$passwort."', '".$telefonnummer."', '".$geschlecht."');";

    $query = $database->prepare ( $sql );
    //print_r($query);
    try {
        $query->execute ();
    } catch ( PDOException $e ) {
        Session::add ( 'response_negative', 'Nutzerkonto konnte nicht angelegt werden.' );
    }
}
```

Abbildung 34: Model Nutzerdaten speichern

Hier werden die Nutzerdaten des Studenten in die Datenbank eingefügt. Falls es einen Fehler geben sollte, wird dieser im „try“ und „catch“ Block abgefangen. In der nächsten Funktion „**saveAddress**“ werden die Adressdaten in die Datenbank übertragen. Die folgende Abbildung zeigt den Quellcode. Diese Funktion befindet sich ebenfalls in der Datei „**UserModel.php**“. Als Parameter werden der Nutzernname, die Hausnummer, die Stadt, das Land und die Postleitzahl aus der View übergeben.

```

public function saveAddress($nutzer, $strasse, $hausnummer, $stadt,
                           $land, $plz) {

    $database = DatabaseFactory::getFactory ()->getConnection ();

    $sql = "INSERT INTO adresse(nutzer_name, straßenname, hausnummer, stadt,
                           land, plz)
           VALUES ('".$nutzer."', '".$strasse."', '".$hausnummer."',
                   '".$stadt."', '".$land."', '".$plz."' );";

    $query = $database->prepare ( $sql );

    try {
        $query->execute ();
    } catch ( PDOException $e ) {
        Session::add ( 'response_negative', 'Adressdaten
                           konnten nicht angelegt werden.' );
    }
}

```

Abbildung 35: Model Adressdaten speichern

Diese Funktion befindet sich ebenfalls in der Datei „**UserModel.php**“. Als Parameter werden der Nutzernname, die Hausnummer, die Stadt, das Land und die Postleitzahl aus der View übergeben. Eine weitere Methode, die in dem Controller aufgerufen wird, ist die Funktion „**saveEmail**“. Hier werden die E-Mail Daten gespeichert.

```

public function saveEmail($nutzer, $email) {

    $database = DatabaseFactory::getFactory ()->getConnection ();

    $sql = "INSERT INTO email(email_name, nutzer_name)
           VALUES ('".$email."', '".$nutzer."');";

    $query = $database->prepare ( $sql );
    //print_r($query);
    try {
        $query->execute ();
    } catch ( PDOException $e ) {
        Session::add ( 'response_negative', 'Email konnte nicht angelegt werden.' );
    }
}

```

Abbildung 36: Model Emaildaten speichern

Wie in **Abbildung 20** dargestellt, werden die E-Mail Daten mit einem INSERT Statement dauerhaft in die Datenbank gespeichert. Hier wird auch ein einheitliches Fehler-Handling verwendet. Die nächste Funktion „**saveStudentData**“ fügt die

spezifischen Daten eines Studenten ein. Diese wären die Studiengangs-ID, die Matrikelnummer, das Fachsemester und das Studiensemester.

```
public function saveStudentData($nutzer, $stid, $matrikel, $fachs, $studs) {  
    ...  
  
    $database = DatabaseFactory::getFactory ()->getConnection ();  
  
    $sql = "INSERT INTO student(nutzer_name, stg_ID, matrikelnummer,  
        fachsemester, studiensemester)  
        VALUES ('".$nutzer."', '".$stid."', '".$matrikel."',  
        '".$fachs."', '".$studs."');"  
  
    $query = $database->prepare ( $sql );  
    //print_r($query);  
    try {  
        $query->execute ();  
        Session::add ( 'response_positive', 'Nutzerkonto für  
            <b>' . $nutzer . '</b> erfolgreich angelegt.' );  
    } catch ( PDOException $e ) {  
        Session::add ( 'response_negative', 'Nutzerdaten konnten  
            nicht angelegt werden.' );  
    }  
}
```

Abbildung 37: Model Studentendaten speichern

Dies sind alle Methoden die nacheinander im Controller aufgerufen werden. Am Schluss wird die Datei „**anlegenStudentView.php**“ nach erfolgreicher Anlage eines Studenten, mit der Funktion aus dem Controller „**zeigeStudentAnlage**“ wieder aufgerufen und ein positives Feedback angezeigt, wie folgt dargestellt.



Abbildung 38: Erfolgreiche Rückgabe bei Anlage eines Studenten

Diese Funktionalität bietet des Weiteren die Möglichkeit, einen Mitarbeiter oder Dozenten anzulegen. Da sich der Quellcode nur in Teilen unterscheidet wird dieser hier nicht weiter ausgeführt.

5.6 Veranstaltung anlegen

Zentraler Bestandteil des Lehrveranstaltungsmanagementsystems sind Veranstaltungen. Diese bestehen aus einer Bezeichnung, dem Kurztext, den Semesterwochenstunden, den Credits, einer Begrenzung für die maximale Anzahl an Teilnehmern, Veranstaltungsart und den Ausstattungsmerkmalen. Zudem kann angegeben werden, ob die Veranstaltung eine Pflichtveranstaltung ist; dafür wird direkt der Studiengang ausgewählt, in dem sie Pflicht ist, zusammen mit dem Semester.

Mithilfe der folgenden Maske können also Veranstaltungen angelegt werden.

Neue Veranstaltung anlegen

Bezeichnung:	<input type="text" value="Programmieren 2"/>
Kurztext:	<input type="text" value="prog2"/>
SWS:	4 <input type="button" value="▼"/>
Credits (ECTS-Punkte):	5 <input type="button" value="▼"/>
maximale Anzahl Teilnehmer:	32 <input type="button" value="▼"/>
Veranstaltungsart:	Vorlesung/Übung <input type="button" value="▼"/>
	<input checked="" type="checkbox"/> Stuhl <input checked="" type="checkbox"/> Computer <input checked="" type="checkbox"/> Beamer <input checked="" type="checkbox"/> Projektor <input type="checkbox"/> Overheadprojektor <input type="checkbox"/> Praesentations-Computer <input type="checkbox"/> HDMI-Anschluss <input type="checkbox"/> Whiteboard <input checked="" type="checkbox"/> Kreidetafel <input type="checkbox"/> Telefon
Ausstattung:	Pflichtvorlesung im Studiengang: Wirtschaftsinformatik, BA <input type="button" value="▼"/>
	<input type="button" value="2"/>
Scheinleistung erforderlich:	ja <input type="button" value="▼"/>
<input style="background-color: #0072bc; color: white; border: 1px solid #0072bc; padding: 5px 10px; border-radius: 5px; font-weight: bold; font-size: 1em; width: fit-content; margin: auto;" type="button" value="anlegen"/>	

Abbildung 39: Ansicht Veranstaltung anlegen

Die Form wird erstellt mit der Methode `anlegenForm()` im „VeranstaltungController“.

5.6.1 Veranstaltung anlegen Form

Controller „Veranstaltung“

```

91*      /*
92   * Diese Methode sammelt die Daten, die für die Anzeige des Formular
93   * notwendig sind, mit dem eine neue Veranstaltung angelegt wird
94   *
95   */
96public function anlegenForm() {
97    $vModel = new VeranstaltungModel ();
98
99    // holt die Veranstaltungsarten aus der Datenbank
100   $vArten = $vModel->getVeranstaltungsarten ();
101
102   // holt die Ausstattung aus der Datenbank
103   $ausstattung = $vModel->getAusstattung ();
104
105   // holt die Studiengänge aus der Datenbank
106   $studiengaenge = $vModel->getStudiengaenge ();
107
108   /*
109    * rendert die Seite
110    * bekommt ein Array mit drei Subarrays übergeben, jeweils eins
111    * für die Veranstaltungsarten, für die Ausstattung und die Studiengänge
112    */
113   $this->View->render ( 'veranstaltung/anlegenForm', array (
114     'vArten' => $vArten,
115     'ausstattung' => $ausstattung,
116     'studiengaenge' => $studiengaenge
117   ) );
118 }
```

Abbildung 40: anlegenForm-Funktion im VeranstaltungController

Unter \$vModel wird ein Objekt vom Veranstaltungsmodel gespeichert. Auf diesem wird die Methode getVeranstaltungsarten() aufgerufen, dessen Rückgabe ein Array mit Veranstaltungsarten samt IDs ist. Mit getAusstattung() werden alle in der Datenbank vorhandenen Ausstattungsobjekte ausgelesen und in \$ausstattung gespeichert. \$studiengaenge speichert alle vorhandenen Studiengänge.

Diese drei Arrays werden dem View anlegenForm übergeben (s.u.)

Model „Veranstaltung“

Mit den folgenden drei Methoden werden Inhalte aus der Datenbank ausgelesen, die in der Maske „Veranstaltung anlegen“ zur Auswahl stehen sollen.

getVeranstaltungsarten() holt alle vorhandenen Veranstaltungsarten aus der Datenbank.

```

400*   /*
401    * Holt alle Veranstaltungsarten aus der Datenbank
402    */
403@ public function getVeranstaltungsarten() {
404    // query-string
405    $q = "SELECT vArt_ID, vArt_bezeichnung from Veranstaltungsart";
406
407    // erzeugt ein Resultset, benutzt dazu die Methode abfrage($q)
408    $result = $this->abfrage ( $q );
409
410    return $result;
411}

```

Abbildung 41: getVeranstaltungsart

```

413*   /*
414    * Holt alle Ausstattungsgegenstände aus der Datenbank
415    */
416@ public function getAusstattung() {
417    // query-string
418    $q = "SELECT ausstattung_ID, ausstattung_bezeichnung from Ausstattung";
419
420    // erzeugt ein Resultset, benutzt dazu die Methode abfrage($q)
421    $result = $this->abfrage ( $q );
422
423    return $result;
424}

```

Abbildung 42: getAusstattung

getAusstattung() holt alle vorhandenen Ausstattungsmerkmale aus der Datenbank.

getStudiengaenge() gibt alle Studiengänge zurück; damit der Studiengangtyp

(Bachelor, Master, etc) dargestellt wird statt der Studiengangtyp-ID, wird ein kleiner Join ausgeführt und die Bezeichnung eingesetzt.

```

447*   /*
448    * Liest alle Studiengaenge aus der Datenbank
449    *
450    */
451@ public function getStudiengaenge() {
452    // query-string
453    $q = "select stg_ID, stg_bezeichnung, stgTyp_kuerzel from Studiengang "
454    . "join Studiengangtyp on Studiengang.stgTyp_ID = Studiengangtyp.stgTyp_ID";
455
456    // erzeugt ein Resultset, benutzt dazu die Methode abfrage($q)
457    $result = $this->abfrage ( $q );
458
459    return $result;
460}

```

Abbildung 43: getStudiengaenge

View „veranstaltung/anlegenForm“:

In der View anlegenForm wird die Form fürs Anlegen einer Veranstaltung zusammengesetzt. Neben festen Elementen wie Bezeichnung und Kurztext gibt es Elemente, für die Daten aus der Datenbank notwendig sind. Diese werden, wie oben beschrieben, über das Model ausgelesen und dann der View als Array übergeben. Zudem gibt es Elemente, für die in einer for-Schleife eine Reihe von

Werten erzeugt wird, bspw. 1-10 für das HTML-Element select für die Semesterwochenstunden.

```

80 <form action=index.php?url=veranstaltung/anlegen method="post">
81
82<table id="veranstaltung-anlegen">
83 <tr><td>Bezeichnung: </td> <td><input type="text" name="veranstaltung_bezeichnung" size="36" maxlength="100"
84 required/></td></tr>
85 <tr><td>Kurztext: </td> <td><input type="text" name="veranstaltung_kurztext" required/></td></tr>
86 <tr><td>SWS: </td> <td>
87
88 <select name="veranstaltung_sws" style="; width: 3em">
89 <?php
90 //befüllt den selection-Abschnitt mit den Werten von 1 bis 10
91 for ($i=1; $i<=10; $i++) {
92 echo "\n\t<option value="" . $i . "\">" . $i . "</option>";
93 }
94 ?>
95 </select>
96 </td></tr>
97 <tr><td>Credits (ECTS-Punkte): </td><td>
98 <select name="veranstaltung_credits" style="; width: 3em">
99 <?php
100 //befüllt den selection-Abschnitt mit den Werten von 1 bis 12
101 for ($i=1; $i<=20; $i++) {
102 echo "\n\t<option value="" . $i . "\">" . $i . "</option>";
103 }
104 ?>
105 </select>
106 </td></tr>
107 <tr><td>maximale Anzahl Teilnehmer: </td><td>
108 <input type="number" min="0" max="500" name="veranstaltung_max_Teilnehmer" size="1" maxlength="3" required/>
109 </td></tr>
110 <tr><td>Veranstaltungsart: </td><td>
111 <select class="input" name="veranstaltung_veranstaltungsart" style="; width: 32em">
112 <?php
113 // erzeugt die Liste mit "option"-HTML-Elementen aus dem Array vArten
114 foreach($this->vArten as $key => $value) {
115 echo "\n\t<option value="" . $value->vArt_ID . "\">";
116 echo utf8_encode($value->vArt_bezeichnung) . "</option>";
117 }
118 ?>
119 </select>
120 </td></tr>
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169

```

Abbildung 44: Veranstaltung anlegen View Teil 1

```

130 <tr><td>Ausstattung: </td>
131 <td>
132<table>
133 <?php
134 // erzeugt eine Tabelle mit allen Ausstattungsmerkmalen|
135 foreach($this->ausstattung as $key => $value) {
136 echo "\n<tr>\n\t<td><input type=\"text\" name=\"veranstaltung_ausstattung[]\" ";
137 echo " size=\"1\" /> </td>";
138 echo "\n\t<td> " . $value->ausstattung_bezeichnung;
139 echo "</td> \n</tr>";
140 }
141 ?>
142 </table>
143 </td></tr>
144 <tr>
145 <td> Pflichtvorlesung im Studiengang: </td>
146 <td>
147 <select class="input" name="veranstaltung_pflichtvorlesung" style="; width: 24em">
148 <?php /*value 0 für kein Pflichtfach*/ ?>
149 <option value="0"> </option>;
150
151 <?php
152 foreach($this->studiengaenge as $key => $value) {
153 echo "\n\t<option value="" . $value->stg_ID . "\">";
154 echo utf8_encode($value->stg_bezeichnung);
155 echo ", " . $value->stgTyp_kuerzel;
156 echo "</option>";
157 }
158 ?>
159 </select>
160 </td></tr>
161 <tr>
162 <td> in Fachsemester: </td><td>
163 <select class="input" name="veranstaltung_fachsemester" style="; width: 3em">
164 <?php
165 for ($i=1; $i<=7; $i++) {
166 echo "\n\t<option value="" . $i . "\">" . $i . "</option>";
167 }
168 ?>
169 </select>

```

Abbildung 45: Veranstaltung anlegen View Teil 2

Der Klick auf „anlegen“ schickt den Formularinhalt per POST an die URL „index.php?url=veranstaltung/anlegen“.

5.6.2 Veranstaltung anlegen

Controller „Veranstaltung“

Daraufhin wird im Controller „Veranstaltung“ die Methode „anlegen“ aufgerufen:

```

120*   /*
121*    * steuert den Anlegen-Vorgang für neue Veranstaltung
122*    *
123*/
124| public function anlegen() {
125    // neues Veranstaltungmodell anlegen
126    $vModel = new VeranstaltungModel ();
127
128    // legt die neue Veranstaltung an und speichert die ID, unter
129    // der sie gespeichert wurde
130
131    if($vID = $vModel->veranstaltungAnlegen ()) {
132
133        if ($vID > 1) {
134
135            //Ausstattung wird hier eingetragen
136            $ausstattung = Request::post('veranstaltung_ausstattung');
137
138            if($vModel->ausstattungEintragen($vID, $ausstattung)) {
139
140                // hole Daten der eben eingetragenen Veranstaltung als ein Array aus Objekten
141                $veranstaltung = $vModel->getVeranstaltung ($vID);
142
143                Session::add ( 'response_positive', 'Neue Veranstaltung erfolgreich angelegt.' );
144
145            } else {
146                Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
147            }
148
149            $this->View->render ( 'veranstaltung/angelegt', array (
150                'veranstaltung' => $veranstaltung
151            ) );
152        } else {
153            Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
154        }
155    } else {
156        Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
157    }
158}

```

Hier wird ein Objekt der Klasse „VeranstaltungModel“ deklariert und instanziert.

Die Methode veranstaltungAnlegen() von diesem Objekt wird ausgeführt (s.u.) und die zurückgegebene Veranstaltungs-ID in die Variable \$vID gespeichert; wenn dieser ein positiver Wert zugewiesen worden ist, wird anschließend die Methode ausstattungEintragen() mit den Parametern \$vID und den Ausstattungsmerkmalen ausgeführt. Wenn alles erfolgreich war, wird die eben eingetragene Veranstaltung aus der Datenbank ausgelesen und der View übergeben und der Session ein repsonse_positive hinzugefügt, damit dieser auf der Folgeseite ausgegeben wird, und die View „veranstaltung/angelegt“ aufgerufen mit der Veranstaltung als Parameter; sonst ein repsonse_negtive mit dem Hinweis, dass ein Fehler aufgetreten ist.

Model „Veranstaltung“

Die Methode veranstaltungAnlegen() trägt die neue Veranstaltung ein. Zudem ruft sie die Methode setPflichtVeranstaltung() auf, die eine Zeile in der Tabelle "Studiengang_hat_Veranstaltung" einträgt, und somit festhält, für welchen Studiengang die Veranstaltung eine Pflichtveranstaltung ist, sofern dies ausgewählt wurde.

Der Rückgabewert von veranstaltungAnlegen() ist die ID, unter der die Veranstaltung in der Datenbank gespeichert wurde, oder -1 falls ein Fehler aufgetreten ist.

```

223@ | public function veranstaltungAnlegen() {
224    $bez = Request::post( 'veranstaltung_bezeichnung' );
225    $kurztext = Request::post( 'veranstaltung_kurztext' );
226    $credits = Request::post( 'veranstaltung_credits' );
227    $sws = Request::post( 'veranstaltung_sws' );
228    $maxNo = Request::post( 'veranstaltung_max_Teilnehmer' );
229    $art = Request::post( 'veranstaltung_veranstaltungsart' );
230    $scheinleistung = Request::post('veranstaltung_scheinleistung');
231    $stg_ID = Request::post( 'veranstaltung_pflichtvorlesung' );
232    $fachsemester = Request::post('veranstaltung_fachsemester');
233    $database = new DatabaseFactoryMysql ();
234
235    $insertString = "INSERT INTO Veranstaltung (`veranst_bezeichnung`, `veranst_kurztext`, "
236    . "`credits`, `SWS`, " . "`maxTeilnehmer`, `vArt_ID`, `scheinleistung` )"
237    . "VALUES ('$bez', '$kurztext', '$credits', '$sws', '$maxNo', '$art', "
238    . "'$scheinleistung');";
239    if ($database->insert( $insertString )) {
240        // speichert die ID, unter der die neue Veranstaltung eingetragen wurde
241        $vID = $database->insert_id;
242
243        // Veranstaltung wird als Pflichtveranstaltung in Tabelle
244        // "Studiengang_hat_Veranstaltung" eingetragen, wenn die gewählte ID > 0
245        if ($stg_ID > 0) {
246            if($this->setPflichtVeranstaltung( $stg_ID, $vID, $fachsemester )) {
247                return $vID;
248            } else {
249                return -1;
250            }
251        }
252        // holt die letzte eingetragene ID
253
254    } else {
255        return -1;
256    }
257}

```

Zunächst werden alle notwendigen Daten aus der Superglobalvariablen `$_POST[]` ausgelesen, dies geschieht über die Hilfsfunktion `Request::Post()`. Unter `$database` wird ein Objekt vom Typ `DatabaseFactoryMysql` gespeichert, das von `Mysqli` erbt. In `$insertString` wird das Sql-Statement fürs Eintragen der neuen Veranstaltung vorbereitet.

Die Methode `getVeranstaltung()` liefert einige Informationen aus der Datenbank bezüglich einer Veranstaltungs-ID. Dazu gehören die Bezeichnung, Kurztext, Credits, SWS, maximale Teilnehmer aber auch Veranstaltungsart, die per Join von einer

Foreign-Key-ID in den zugehörigen Text verwandelt wird.

```

384     // holt die veranstaltung mit id = $vID
385     public function getVeranstaltung($vID) {
386
387         $q = "select Veranstaltung.veranst_ID, Veranstaltung.veranst_bezeichnung, Veranstaltung.veranst_kurztext, "
388             . "Veranstaltung.credits, Veranstaltung.SWS, "
389             . "Veranstaltung.maxTeilnehmer, Veranstaltungsart.vArt_bezeichnung as Veranstaltungsart "
390             . "from Veranstaltung join Veranstaltungsart on Veranstaltung.vArt_ID = Veranstaltungsart.vArt_ID"
391             . " where Veranstaltung.veranst_ID = $vID;";
392
393         $result = $this->abfrage ( $q );
394
395         return $result;
396     }

```

View „veranstaltung/angelegt“

```

45  <?php
46      $table = new Table ();
47      // Tabelle drucken mit alias
48      $alias = array (
49          "0" => "Veranstaltungsnummer",
50          "1" => "Bezeichnung",
51          "2" => "Kurztext",
52          "3" => "Credits",
53          "4" => "SWS",
54          "5" => "maximale Anzahl Teilnehmer",
55          "6" => "Veranstaltungsart"
56      );
57      $alias = ( object ) $alias;
58      $table->table ( array (
59          'table' => $this->veranstaltung,
60          'alias' => array (
61              $alias
62          )
63      ) );
64 ?>

```

In der View wird die übergebene Veranstaltung in einer generischen Tabelle ausgegeben.

Das Ergebnis sieht dann so aus:

The screenshot shows a web page with a green header bar containing the text "Neue Veranstaltung erfolgreich angelegt.". Below this is a white content area with a title "Veranstaltung erfolgreich angelegt". Underneath the title is a table with the following data:

Veranstaltungsnummer	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart
35	Programmieren 2 prog2	5	4	32		Vorlesung/Übung

5.7 Räume anlegen

Die User Story zu dieser Funktionalität lautet: „Als Mitarbeiter möchte ich Räume anlegen können. (Nr. 110)“. Einem als Mitarbeiter eingeloggter User soll es also möglich sein Räume jeder Art anzulegen. Bei den Räumen unterscheidet man zwischen Vorlesungsräumen, Laborräumen, Büros und Bibliotheken. Diese sind mit dem Konzept der Spezialisierung im ER-Model umgesetzt. Jeder Raum hat dabei Stammdaten, bestehend aus einer Bezeichnung und einem Gebäude, in dem der

Raum sich befindet. Auch die im Raum vorhandene Ausstattung, welche für Vorlesungs- und Laborräume wichtig ist, soll mit angelegt werden können. Nun zur Umsetzung dieser Idee.

Zunächst muss ausgewählt welche Kategorie von Raum angelegt werden soll. Dazu ein einfaches Auswahlverfahren mit Radiobuttons.

```
<article>
    <h1>Raumkategorie auswählen</h1>
    <!-- raumSelected-function des Controllers wird ausgeführt nach der Bestätigung -->
    <form action="index.php?url=raumAnlegen/raumSelected" method="post" />
        <input type="radio" name="typ" id="vorlesungsraum"
            value="vorlesungsraum">Vorlesungsraum<br> <input type="radio"
            name="typ" id="buero" value="buero">Büro<br> <input type="radio"
            name="typ" id="labor" value="labor">Labor<br> <input type="radio"
            name="typ" id="bibliothek" value="bibliothek">Bibliothek<br>
        <p>
            <input class="button" type="submit" id="b1" value="weiter"></a>
        </p>
    </form>
</article>
```

Abbildung 46: raumAnlegenView

Daraus ergibt sich folgender Inhalt für die Startseite.

Raumkategorie auswählen

- Vorlesungsraum
- Büro
- Labor
- Bibliothek

weiter

Abbildung 47: Startseite Raum anlegen

Beim klicken auf den „weiter“-Button wird im Controller die raumSelected-Funktion aufgerufen. Diese fragt durch die Post-Methode ab, welcher Radiobutton ausgewählt wurde und zeigt daraufhin die entsprechende View an. Im folgenden Bild ist ab dem Fall „labor“ abgeschnitten. Die Zeilen für Laborräume und Bibliotheken entsprechen derselben Logik und lassen dementsprechende Views anzeigen.

```

public function raumSelected() {
    if (isset ( $_POST['raumtyp'])) {
        $raumtyp = $_POST ['raumtyp'];
        switch ($raumtyp) {
            case "vorlesungsraum" :
                $this->View->render ( 'raumanlegen/vorlesungsraumAnlegen', array (
                    'geb_list' => raumAnlegenModel::getGebäude (),
                    'ausstattung_list' => raumAnlegenModel::getAusstattung ()
                ) );
                break;
            case "buero" :
                $this->View->render ( 'raumanlegen/bueroAnlegen', array (
                    'geb_list' => raumAnlegenModel::getGebäude ()
                ) );
                break;
            case "labor" :

```

Abbildung 48: raumSelected-Funktion raumAnlegenController

Für den Fall, dass ein Fehler auftritt oder fälschlicher Weise kein Raumtyp ausgewählt wurde, wird dementsprechend eine Fehler- oder Warnungsmeldung ausgegeben.

```

        default :
            Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
        }
    } else {
        Session::add ( 'response_warning', 'Es ist kein Raumtyp ausgewählt.' );
        $this->View->render ( 'raumanlegen/raumAnlegen' );
    }
}

```

Abbildung 49: raumSelected-Funktion raumAnlegenController Fehlerbehandlung

Nun wurde also in der Anwendung ein Raumtyp ausgewählt. Die angezeigten Views entsprechen folgenden Darstellungen für die Varianten Vorlesungsraum, Laborraum, Büro und Bibliothek.

<h3>Vorlesungsraum anlegen</h3> <p>Bezeichnung <input type="text"/></p> <p>Gebäude <input type="text" value="1. Prittitzstraße , 10"/> ▾</p> <p>Stuhl <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Computer <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Beamer <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Projektor <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Overheadprojektor <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Praesentations-Computer <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>HDMI-Anschluss <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Whiteboard <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Kreidetafel <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Telefon <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Zurück Vorlesungsraum anlegen</p>	<h3>Labor/Werkstatt anlegen</h3> <p>Bezeichnung <input type="text"/></p> <p>Gebäude <input type="text" value="1. Prittitzstraße , 10"/> ▾</p> <p>Laborat <input type="text" value="1. Rechnerlabor"/> ▾</p> <p>Stuhl <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Computer <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Beamer <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Projektor <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Overheadprojektor <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Praesentations-Computer <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>HDMI-Anschluss <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Whiteboard <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Kreidetafel <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Telefon <input type="checkbox"/> Anzahl: <input type="text"/></p> <p>Zurück Labor anlegen</p>
---	---

Abbildung 50: Darstellung vorlesungsraumAnlegenView Abbildung 51: Darstellung laborraumAnlegenView

<h3>Bibliothek anlegen</h3> <p>Bezeichnung <input type="text"/></p> <p>Gebäude <input type="text" value="1. Prittitzstraße , 10"/> ▾</p> <p>Folgende Buchkategorien liegen in der Bibliothek vor:</p> <ul style="list-style-type: none"> Astronomie <input type="checkbox"/> Physik <input type="checkbox"/> Literatur <input type="checkbox"/> Mathematik <input type="checkbox"/> Medizin <input type="checkbox"/> Physik <input type="checkbox"/> Wirtschaftsinformatik <input type="checkbox"/> Wirtschaftswissenschaften <input type="checkbox"/> <p>Zurück Bibliothek anlegen</p>	<h3>Büro anlegen</h3> <p>Bezeichnung <input type="text"/></p> <p>Gebäude <input type="text" value="1. Prittitzstraße , 10"/> ▾</p> <p>Zurück Büro anlegen</p>
---	---

Abbildung 52: Darstellung büroAnlegenView

Abbildung 53: Darstellung bibliothekAnlegenView

Die Views beziehen dabei alle die raumStammdatenView mit ein. Diese sieht wie folgt aus. Als Beispiel hierfür sieht man im folgenden Bild die vorlesungsraumAnlegenView, welche die raumStammdatenView und die raumAusstattungView einbezieht.

```
<article>
    <h1>Vorlesungsraum anlegen</h1>
    <!-- saveRaum-function wird ausgeführt nach der Bestätigung -->
    <form action="index.php?url=raumAnlegen/saveRaum" method="post">
        <!-- inkludet das Textfeld für die Bezeichnungseingabe und die Wahl des Gebäudes -->
        <?php
            include 'raumStammdatenView.php';
        ?>
        <!-- inkludet Ausstattungsabfrage -->
        <?php
            include 'raumAusstattungView.php';
        ?>
        <input type="hidden" name="raumtyp" value="vorlesungsraum">
    <p>
        <?php
            echo '<input class="button" type="button" value="Zurück" onClick="history.back() ;" >';
        ?>
        <input class="button" type="submit" name="anlegen" value="Vorlesungsraum anlegen">
    </p>
</form>
</article>
```

Abbildung 54: vorlesungsraumAnlegenView

Die raumStammdatenView wird durch folgenden Quellcode erstellt.

```
// Für die Stammdaten, die jeder Raum hat
echo '<table style="border: 0px; text-align: left;"><tr>';
// Bezeichnung im Textfeld eingeben.
echo "<th>Bezeichnung</th>";
echo '<th><input class="tf" type = "text" name = "bezeichnung"></th></tr>';
// Alle Gebäude werden in einer Liste zur Auswahl angezeigt. Mit Bezeichnung und Anschrift.
echo "<tr> <th>Gebäude</th>";
echo '<th><select name="gebäude">';
if ($this->geb_list) {
    foreach ( $this->geb_list as $key => $value ) {
        echo "<option>";
        echo htmlentities ( $value->geb_bezeichnung );
        echo " , ";
        echo htmlentities ( $value->straßenname );
        echo " , ";
        echo htmlentities ( $value->hausnummer );
        echo "</option>";
    }
} else {
    Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
}
echo '</select></th></tr></table>';
```

Abbildung 55: raumStammdatenView

Diese besteht aus einem Textfeld für die Bezeichnung und einem Dropdown-Menü, welches die Gebäude auflistet. Hier wird also das Gebäude ausgewählt, in dem der Raum sich befindet. Die zur Auswahl stehenden Gebäude werden der View als Array

übergeben. Das Array ergibt sich aus der getGebäude-Funktion des raumAnlegenModels.

```
// Alle Informationen von Gebäuden abfragen.
public function getGebäude() {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $sql = 'Select g.geb_bezeichnung, a.strassenname, a.hausnummer from Gebaeude g
            join Adresse a on g.geb_bezeichnung = a.geb_bezeichnung';
    $query = $database->prepare ( $sql );
    $query->execute ();
    return $query->fetchAll ();
}
```

Abbildung 56: getGebäude raumAnlegenModel

Nach demselben Prinzip stellen sich auch die Wahloption der Laborart in der laborraumAnlegen-View dar. Eine getLaborart-Funktion im raumAnlegenModel besorgt die Informationen über die möglichen Laborarten. Der View werden diese als Array übergeben und diese stellt sie als Dropdown-Menü dar.

Des Weiteren beziehen die vorlesungsraumAnlegenView (siehe Beispiel) und die laborraumAnlegenView die raumAusstattungView mit ein. Diese stellt die in der Datenbank vorliegenden Ausstattungen dar. Dies passiert wie gesehen durch die Ausstattungsbezeichnung, eine Checkbox und ein Textfeld für die Anzahl. Abgefragt wird nach der Ausstattung durch die getAusstattung-Funktion des raumAnlegenModels.

```
// Alle Informationen von Tabelle Ausstattung abfragen.
public function getAusstattung() {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $sql = 'Select ausstattung_ID, ausstattung_bezeichnung from Ausstattung';
    $query = $database->prepare ( $sql );
    $query->execute ();
    return $query->fetchAll ();
}
```

Abbildung 57: getAusstattung raumAnlegenModel

Die Darstellung der daraus gewonnenen und als Array übergebenen Datensätze erfolgt in der raumAusstattungView. Die Darstellung wird in Form einer Tabelle realisiert.

```
$data_ausstattung = array ();
echo '<table style="border: 0px; text-align: left; font-weight: lighter;">';
if ($this->ausstattung_list) {
    foreach ( $this->ausstattung_list as $key => $value ) {
        echo '<tr><th>';
        echo htmlentities ( $value->ausstattung_bezeichnung );
        echo '</th><th><input type="checkbox" name=""';
        echo htmlentities ( $value->ausstattung_bezeichnung );
        echo '" value=""';
        echo htmlentities ( $value->ausstattung_ID );
        echo '></th><th>Anzahl: <input class="tf" type="text" name = ""';
        echo htmlentities ( $value->ausstattung_bezeichnung );
        echo 'Anzahl"/></th></tr>';
        array_push ( $data_ausstattung, htmlentities ( $value->ausstattung_bezeichnung ) );
    }
}
echo '</table>';
} else {
    Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
}
```

Abbildung 58: raumAusstattungView

Für jeden Datensatz im Array wird also eine Tabellenreihe erzeugt, die demensprechen mit Inhalt gefüllt ist. So kann dynamisch nach der Ausstattung abgefragt werden. Wichtig sind hierbei die Namen der Checkboxen und Textfelder. Denn diese werden später verwendet um die Informationen in die Datenbank einzutragen.

Für den Fall, dass eine Bibliothek angelegt werden soll, ist die Möglichkeit geboten verschiedene Buchkategorien zu wählen, welche in der Bibliothek vorliegen könnten. Hierzu werden ähnlich wie bei der Ausstattung Bezeichnungen mit Checkboxen erstellt. Die Datensätze kommen hierfür von der getBuchkategorien-Funktion.

```
// Alle Buchkategorien abfragen.
public function getBuchkategorie() {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $sql = 'Select buchKat_ID, buchKat_bezeichnung from Buchkategorie';
    $query = $database->prepare ( $sql );
    $query->execute ();
    return $query->fetchAll ();
}
```

Abbildung 59: getBuchkategorien raumAnlegenModel

```
Folgende Buchkategorien liegen in der Bibliothek vor: <br>
<?php
if ($this->buchKat_list) {
    echo "<table>";
    foreach ( $this->buchKat_list as $key => $value ) {
        echo '<tr><th>';
        echo htmlentities ( $value->buchKat_bezeichnung );
        echo '</th><th><input type="checkbox" name=""';
        echo htmlentities ( $value->buchKat_bezeichnung );
        echo '" value="1"></th>';
        echo '</tr>';
    }
    echo "</table>";
} else {
    echo "Es sind keine Buchkategorien in der Datenbank.";
}
?>
```

Abbildung 60: bibliothekAnlegenView Darstellung Buchkategorien

Damit ist nun für jeden Raumtyp gezeigt wie die jeweiligen Views zusammengesetzt sind. Nun geht es darum die eingegebenen Attribute eines solchen Raums in der Datenbank zu speichern. Hierzu gibt es im Controller die saveRaum-Funktion, die von jeder der Views aufgerufen wird, sobald der jeweilige „Raum anlegen“-Button. Im Folgenden nur bis Ende des Falles Vorlesungsraum angezeigt. Für die anderen Fälle verfährt der Controller nach demselben Prinzip. Es werden lediglich die verschiedenen Informationen berücksichtigt.

```
public function saveRaum() {
    if (isset ( $_POST ['bezeichnung'] ) and isset ( $_POST ['gebäude'] ) and isset ( $_POST ['raumtyp'] )) {
        // fragt die übergebenen Stammdaten ab
        $bezeichnung = $_POST ['bezeichnung'];
        $gebäude_string = $_POST ['gebäude'];
        $gebäude_array = explode ( ", ", $gebäude_string );
        $gebäude_char = $gebäude_array [0];
        $raumtyp = $_POST ['raumtyp'];

        // legt den Raum (Stammdaten in Raum-Tabelle ) in der Datenbank an.
        raumAnlegenModel::raumStammdaten ( $bezeichnung, $gebäude_char );

        switch ( $raumtyp ) {
            case "vorlesungsraum" :
                // Raum wird in die Tabelle der jeweiligen Raumkategorie eingetragen.
                raumAnlegenModel::vorlesungsraumAnlegen ( $bezeichnung );
                /*
                * Um die angegebene Ausstattung für den Vorlesungsraum in die Datenbanktabelle einzutragen,
                * werden alle Checkboxen und dazugehörigen Textfelder für die Anzahl abgefragt.
                * Die Informationen werden dann mit der ausstattungVorlesungsraum-funktion im Model in die Datenbank eingetragen.
                */
                $data = array (
                    'ausstattung_list' => raumAnlegenModel::getAusstattung ()
                );
                // print_r($data);
                foreach ( $data as $key => $value ) {
                    $this->($key) = $value;
                    // print_r($value);
                }
                foreach ( $this->ausstattung_list as $key => $value ) {
                    // echo htmlentities($value[0]);
                    // print_r($value->ausstattung_bezeichnung);
                    if (isset ( $_POST [$value->ausstattung_bezeichnung] )) {
                        $id = $value->ausstattung_ID;
                        $anzahl = $_POST [$value->ausstattung_bezeichnung . "Anzahl"];
                        raumAnlegenModel::ausstattungVorlesungsraum ( $id, $bezeichnung, $anzahl );
                    }
                }
            break;
        }
    }
}
```

Abbildung 61: saveRaum-Funktion raumAnlegenController

Nun zum Zweck der saveRaum-Funktion. Dieser besteht darin, die Eingaben des Users zu verwerten und in der Datenbank zu sichern. Das bedeutet zunächst werden die Stammdaten verwertet und durch die raumStammdaten-Funktion im Model gespeichert.

```
public function raumStammdaten($bezeichnung, $gebäude_char) {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $insert_sql = "INSERT INTO raum(raum_bezeichnung, geb_bezeichnung)
                  VALUES ('$bezeichnung', '$gebäude_char')";
    $query1 = $database->prepare ( $insert_sql );
    try{
        $query1->execute ();
    } catch (PDOException $e){
        Session::add ( 'response_negative', 'Der Raum konnte nicht angelegt werden.' );
        $this->View->render ( 'raumanlegen/raumAngelegt' );
        exit;
    }
}
```

Abbildung 62: raumStammdaten-Funktion im raumAnlegenModel

Für den Fall das hier ein Fehler entsteht, zum Beispiel durch eine doppelte Bezeichnung in der Datenbank, wird hier abgebrochen.

Des Weiteren müssen für einen Vorlesungsraum die Bezeichnung in die Vorlesungsraum-Tabelle und die Ausstattung in die Vorlesungsraum_hat_Ausstattung-Tabelle eingetragen werden. Für das Beispiel des Vorlesungsraums wird der Ablauf nun genauer erklärt. Die Funktion, die den Raum in die Vorlesungsraum-Tabelle einträgt, lautet wie im Controller zu sehen vorlesungsraumAnlegen.

```
// Vorlesungsraum in Datenbank eintragen
public function vorlesungsraumAnlegen($bezeichnung) {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $insert_sql2 = "INSERT INTO vorlesungsraum (raum_bezeichnung)
                    VALUES ('$bezeichnung')";
    $query2 = $database->prepare ( $insert_sql2 );
    try{
        $query2->execute ();
        Session::add ( 'response_positive', 'Der Vorlesungsraum wurde erfolgreich angelegt.' );
    } catch (PDOException $e){
        Session::add ( 'response_negative', 'Der Vorlesungsraum konnte nicht angelegt werden.' );
    }
}
```

Abbildung 63: vorlesungsraumAnlegen-Funktion im raumAnlegenModel

Die Ausstattung abzuspeichern ist etwas komplizierter. Durch eine erneute Abfrage der Ausstattungstabelle liegen alle Informationen über mögliche Ausstattungen vor. Dieses Array \$data wird genutzt um die Checkboxen der View durchzugehen. Ist eine Checkbox gesetzt wird die dazugehörige Ausstattungs-ID genommen und mit der entsprechenden Anzahl aus dem Textfeld in die Vorlesungraum_hat_Ausstattung-Tabelle eingetragen. Dies wird von der ausstattungVorlesungsraum-Funktion des Models übernommen.

```
public function ausstattungVorlesungsraum($id, $bezeichnung, $stückzahl) {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $insert_sql = "INSERT INTO vorlesungsraum_hat_ausstattung (ausstattung_ID, raum_bezeichnung, anzahl)
                    VALUES ('$id','$bezeichnung', '$stückzahl')";
    $query = $database->prepare ( $insert_sql );
    try{
        $query->execute ();
    } catch (PDOException $e){
        Session::add ( 'response_negative', 'Es ist ein Fehler beim Eintragen der Ausstattung aufgetreten.' );
    }
}
```

Abbildung 64: ausstattungVorlesungsraum-Funktion im raumAnlegenModel

Für ein Laborraum gibt es größtenteils dieselben Informationen zum abspeichern. Stammdaten, sowie die Ausstattung sind hier ebenfalls vorhanden und werden gleich abgespeichert. Zusätzlich wird dazu noch die Laborart eingetragen. Die laborAnlegen-Funktion übernimmt diese Aufgabe.

Die Bibliothek hat neben den Stammdaten noch die Buchkategorien zum eintragen in die Datenbank. Hierzu gibt es die buchKatBibliothek-Funktion im Model.

Für ein Büro müssen nur die Stammdaten eingetragen werden.

Damit sind die verschiedenen Raumtypen angelegt und die Funktionalität des Raum anlegen hergestellt. Das erfolgreiche Anlegen eines Raums wird dann wie folgt dargestellt.



Abbildung 65: Ansicht - Vorlesungsraum erfolgreich angelegt

5.8 Stundenplan anzeigen

Als Student und Dozent kann für jedes Fachsemester eines beliebigen Studiengangs ein Stundenplan angezeigt werden.

Des Weiteren ist es möglich, sich einen individuellen Stundenplan anzeigen zu lassen. In der Rolle als Student kann demnach ein Studentenstundenplan abgerufen werden, und in der Rolle als Dozent ein Dozentenstundenplan.

Ähnlich wie bei der Anzeige der Veranstaltungstermin-Übersicht wird die Anzeige des Stundenplans aufgebaut, entscheidender Unterschied ist, dass aus der Session ausgelesen werden muss, welche Rolle der angemeldete Benutzer besitzt, um so sicherzustellen, ob ein Studentenstundenplan oder ein Dozentenstundenplan generiert werden soll.

Im Controller wird daher beim Aufruf der Model-Methode **getVeranstaltungstermine_individuell()** der Nutzername und die Rolle des angemeldeten Nutzers mit Session-Variablen übergeben:

```
//neue Model-Objekte für Datenfluss erstellen:
$spModel = new stundenplanModel();
$rzModel = new raumZuweisenModel();

//Daten aus DB holen:
$veranstaltungstermine = $spModel->getVeranstaltungstermine_individuell(Session::get('user_name'), Session::get('user_role'));
$wochentage = $rzModel->getWochentage();
$stundenZeiten = $rzModel->getStundenzeiten();
$name = $spModel->getName(); //Name des Nutzers (Student oder Dozent)

$this->View->render('stundenplan/stundenplanIndividuell',
array('veranstaltungstermine' => $veranstaltungstermine,
      'wochentage' => $wochentage,
      'stundenzeiten' => $stundenZeiten,
      'name' => $name));
```

Abbildung 66: stundenplanController, Funktion stundenplan_individuell()

Im Model wurde die Methode **getVeranstaltungstermine_individuell()** folgendermaßen implementiert:

```

public function getVeranstaltungstermine_individuell($nutzer_name, $rolle_ID) {
    if($rolle_ID == 1) { //Nutzer hat die Rolle Student
        $query = "SELECT vt.tag_ID, vt.stdZeit_ID, vt.raum_bezeichnung, v.veranst_kurztext, n.nachname, stg.stg_kurztext,
                  shv.pflicht_im_Semester
                  FROM Veranstaltungstermin vt
                  JOIN Veranstaltung v ON (vt.veranst_ID = v.veranst_ID)
                  JOIN Student_angemeldetAn_Veranstaltung sav ON (v.veranst_ID = sav.veranst_ID)
                  LEFT JOIN Dozent doz ON (v.dozent_nutzer_name = doz.nutzer_name)
                  LEFT JOIN Nutzer n ON (doz.nutzer_name = n.nutzer_name)
                  JOIN Studiengang_hat_Veranstaltung shv ON (v.veranst_ID = shv.veranst_ID)
                  JOIN Studiengang stg ON (shv.stg_ID = stg.stg_ID)
                  WHERE sav.student_nutzer_name = '$nutzer_name'
                  ORDER BY vt.stdZeit_ID, vt.tag_ID;";

        $veranstaltungstermine = $this->abfrage($query);
    }
    else if($rolle_ID == 2) { //Nutzer hat die Rolle Dozent
        $query = "SELECT vt.tag_ID, vt.stdZeit_ID, vt.raum_bezeichnung, v.veranst_kurztext, stg.stg_kurztext, shv.pflicht_im_Semester
                  FROM Veranstaltungstermin vt
                  JOIN Veranstaltung v ON (vt.veranst_ID = v.veranst_ID)
                  JOIN Studiengang_hat_Veranstaltung shv ON (v.veranst_ID = shv.veranst_ID)
                  JOIN Studiengang stg ON (shv.stg_ID = stg.stg_ID)
                  JOIN Dozent doz ON (v.dozent_nutzer_name = doz.nutzer_name)
                  WHERE v.dozent_nutzer_name = '$nutzer_name'
                  ORDER BY vt.stdZeit_ID, vt.tag_ID;";

        $veranstaltungstermine = $this->abfrage($query);
    }
    return $veranstaltungstermine;
}

```

Abbildung 67: studenplanModel, Funktion getVeranstaltungstermine_individuell()

Besitzt der Benutzer die Rolle Student, so werden alle Veranstaltungstermine als Resultset in einem Array zurückgegeben, an welchen der Student angemeldet ist.

Besitzt er jedoch die Rolle Dozent, so werden alle Veranstaltungstermine als Resultset in einem Array zurückgegeben, welche der Dozent leitet.

Die folgende Abbildung zeigt die Ansicht eines Studentenstundenplans:

Stundenplan für Max Mustermann:						
Stundenzeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag
08:00:00 - 09:30:00	WFPRJ • WF5 • A100 • Klippel	DAWA • WF5 • A100 • Herbort	FUEBIT • WF5 • A100 • Waldert	IKOM • WF5 • V101 • Reisach	OR • WF5 • A100 • Gramlich	
09:50:00 - 11:20:00	WFPRJ • WF5 • A100 • Klippel	DAWA • WF5 • A100 • Herbort	FUEBIT • WF5 • A100 • Waldert		OR • WF5 • A100 • Gramlich	
11:30:00 - 13:00:00	WFPRJ • WF5 • A100 • Klippel			IKOM • WF5 • V101 • Reisach		
14:00:00 - 15:30:00				IKOM • WF5 • V101 • Reisach	ITA • WF5 • V101 • Gewald	
15:45:00 - 17:15:00	SEMWF • WF5 • A100 • Grieble				ITA • WF5 • F007 • Gewald	
17:25:00 - 18:55:00						
19:00:00 - 20:30:00						

Abbildung 68: Studentenstundenplan

5.9 Raumplan anzeigen

Ein Raumplan kann von allen Rollen (Mitarbeiter, Student und Dozent) abgerufen werden.

Der Raumplan wird ähnlich generiert wie der Stundenplan.

Zuallererst muss der Benutzer den Raum auswählen, von welchem der Raumplan angezeigt werden soll:



Abbildung 69: Formular zur Auswahl eines Raumes

Im Controller werden dazu alle Räume über das Model geholt, indem die Methoden **getVorlesungsräume()** und **getLaborräume()** aufgerufen werden. Mit **render()** werden dann die zur Anzeige erforderlichen Daten an den View weitergegeben, sodass dem Benutzer alle Räume zur Auswahl angezeigt werden können:

```
//neues Model-Objekt für Datenfluss erstellen:  
$rpModel = new raumplanModel();  
  
//Daten aus DB holen:  
$vorlesungsräume = $rpModel->getVorlesungsräume();  
$laborräume = $rpModel->getLaborräume();  
  
$this->View->render('raumplan/formular', array('vorlesungsräume' => $vorlesungsräume, 'laborräume' => $laborräume));
```

Abbildung 70: raumplanController, Funktion erzeugeFormular()

Wurde ein Raum ausgewählt und das Formular abgeschickt, so wird der ausgewählte Raum abgespeichert und als Parameter beim Aufruf der Methode

getVeranstaltungstermine() übergeben, welche alle Veranstaltungstermine liefert, die im ausgewählten Raum stattfinden.

Des Weiteren müssen die Stundenzeiten und Wochentage über das Model geholt werden, sodass ein Raumplan generiert werden kann.

Mit **render()** werden die geholten Daten dann an die View weitergeleitet.

Folgende Abbildung zeigt den dazugehörigen Code:

```
//Formularverarbeitung:
$raum_bezeichnung = $_POST['waehleRaum'];

//neue Model-Objekte für Datenfluss erstellen:
$rpModel = new raumplanModel();
$rzModel = new raumZuweisenModel();

//Daten aus DB holen:
$veranstaltungstermine = $rpModel->getVeranstaltungstermine($raum_bezeichnung);
$wochentage = $rzModel->getWochentage();
$stundenZeiten = $rzModel->getStundenzeiten();

$this->View->render('raumplan/ausgabe', array('raum' => $raum_bezeichnung,
                                                'veranstaltungstermine' => $veranstaltungstermine,
                                                'wochentage' => $wochentage,
                                                'stundenzeiten' => $stundenZeiten));
```

Abbildung 71:raumplanController, Funktion raumplanAnzeigen()

Die Ansicht des Raumplans sieht dann folgendermaßen aus:

Raumbelegung V001:						
Stundenzeit	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag
08:00:00 - 09:30:00				SOFEN ‣ WF4 ‣ Graf	BSYS ‣ WF3 ‣ Hasbargen	
09:50:00 - 11:20:00					DAPRO ‣ WF4 ‣ KlippeL	
11:30:00 - 13:00:00					INFM ‣ WF4 ‣ Lang	
14:00:00 - 15:30:00						
15:45:00 - 17:15:00			BTENG ‣ WF3 ‣ Halton			
17:25:00 - 18:55:00					MARK ‣ WF3 ‣ Reisach	
19:00:00 - 20:30:00						

[zurück](#)

Abbildung 72: Raumplan

5.10 Teilnehmerlisten für Dozenten anzeigen

Eine weitere hilfreiche Funktion, die bei diesem Projekt in einer User Story instanziert wurde, beschreibt die Anzeige von Teilnehmerlisten für einen Dozenten. Hierzu wurde folgende User Story formuliert:

„Nr.: xxx, Als Mitarbeiter möchte ich div. Nutzer anlegen können“

Diese User Story wurde ebenfalls gemäß des MVC Patterns entwickelt. Zunächst soll der Output des fertigen Programms in folgender Grafik dargestellt werden.

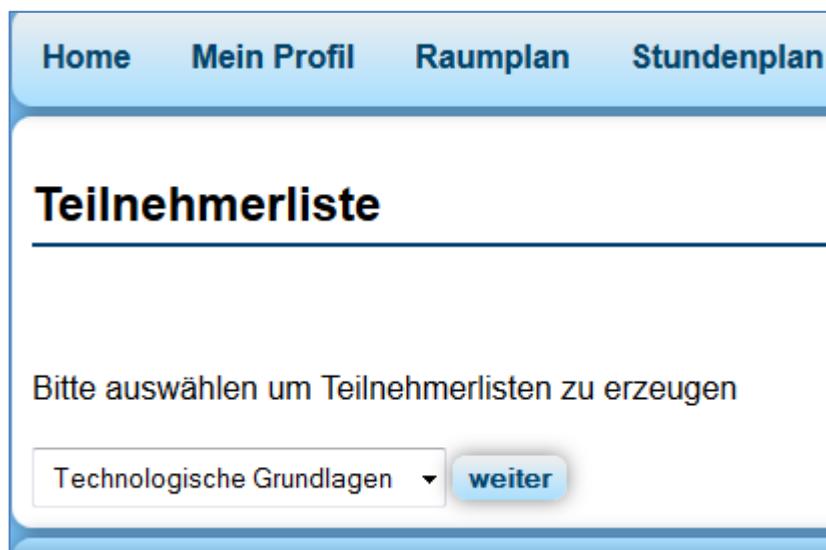


Abbildung 73: View Auswahl einer Vorlesung

Hier sieht man, dass das Dropdown Feld bereits mit Werten gefüllt wurde. Diese Werte wurden von dem Controller im Model aufgerufen. Die Funktionen befinden sich in den Dateien „**DozentController.php**“ und „**DozentModel.php**“. Wenn der Button „weiter“ geklickt wird, erscheint die Teilnehmerliste für die ausgewählte Vorlesung. Diese Anzeige wird in der folgenden Grafik dargestellt.

Teilnehmerliste für Kurs "Technologische Grundlagen"

Nutzername	Email	Matrikel	Vorname	Nachname	Telefonnummer	Studiengang	Bezeichnung	
cadoll	cadoll@hs-ulm.de	3111111	Carson	Doll		Wirtschaftsinformatik	Technologische Grundlagen	Email verschicken
doritc	doritc@hs-ulm.de	5111111	Dolan	Ritch		Wirtschaftsinformatik	Technologische Grundlagen	Email verschicken
loluke	loluke@hs-ulm.de	4111111	Logan	Luke		Wirtschaftsinformatik	Technologische Grundlagen	Email verschicken
modoof	modoof@hs-ulm.de	2111111	Moritz	Doof		Wirtschaftsinformatik	Technologische Grundlagen	Email verschicken
Email an Alle								
Zurück Auswahl								

copyright und andere informationen über diese seite

Abbildung 74: View Teilnehmerliste für die Vorlesung TEGR

Hier werden beispielsweise alle Nutzer des Studiengangs Technologische Grundlagen angezeigt.

5.11 Profile anzeigen lassen

Nachdem eine Grundlage für die Zusammenarbeit mit mehreren Entwicklern geschaffen war, konnte das Lehrveranstaltungssystem nun mit Funktionen ausgestattet werden. In diesem Teilkapitel wird, die Entwicklung und Herangehensweise der Funktion „Profile anzeigen lassen“ beleuchtet. In den meisten Rich-Client Applikationen gehört diese Funktion zum Standard. Daher wurde zunächst folgende User Story formuliert:

„Nr.: xxx, Als Dozent möchte ich mir mein Profil anzeigen lassen und bearbeiten können“

Zunächst soll das Ergebnis der Aufgabe dargestellt werden. Hierzu wurde die Datei meinProfilView.php erstellt, die sich im Ordnerpfad application/view/dozent befindet. In **Abbildung 2** ist die Browserseite zum Anzeigen und Bearbeiten des Profils eines Dozenten abgebildet. Diese User Story wurde auch für die Rollen Mitarbeiter und Student zur Verfügung gestellt. Der Quellcode wird aber nur für die Rolle Dozent näher erläutert, da er sich für die anderen beiden Rollen ähnlich verhält.

The screenshot shows a web application interface for managing a professor's profile. At the top, there is a blue header bar with navigation links: Home, Mein Profil, Raumplan, Stundenplan, and Teilnehmerliste. Below the header, the title "Mein Profil" is displayed in bold. The main content area contains a table of personal information with edit buttons:

Nutzername	dozent	
Vorname	Klaus	
Nachname	Kopf	Ändern
Straße	Schaffnerstraße	Ändern
Hausnummer	10	Ändern
PLZ	89073	Ändern
Stadt	Ulm	Ändern
Land	Deutschland	Ändern
Telefonnummer	0731 784514	Ändern
Email	dozent@hs-ulm.de	Ändern

Below the table, there is a button labeled "Daten Neu Laden". At the bottom right of the page, the text "copyright und and" is visible.

Abbildung 75: Profil eines Dozenten anzeigen und bearbeiten

Entsprechend des MVC Patterns gehört die Datei zur View Komponente. Hier kann jeder eingeloggte Dozent seine Personalien ändern bzw. anzeigen lassen. Die Felder Nutzername und Vorname sind nur im Anzeigemodus möglich, da der Nutzername generisch bei Anlage definiert wird und der Vorname sich in der Regele in den seltensten Fällen ändert. Über den Button „**Daten Neu Laden**“ können die Daten nochmal aktualisiert werden. Der Button „**Ändern**“ hinter jedem Attribut ändert das entsprechende Feld dauerhaft in der Datenbank. Da bereits die Umgebung der Applikation anhand des MVC Pattern realisiert wurde, hat es sich angeboten diese und alle anderen User Stories objektorientiert und anhand des MVC-Patterns zu entwickeln. Das MVC Pattern wurde in folgende Dateistruktur unterteilt:

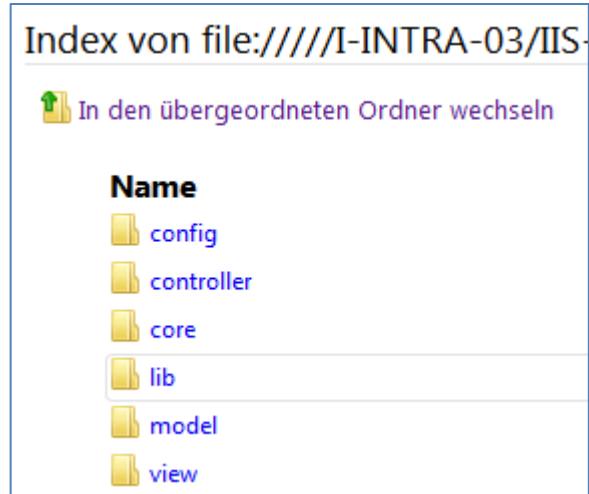


Abbildung 76: Ordnerstruktur für das Projekt

Hier kann man die Ordner für die View, den Controller und das Model erkennen. Im Model-Verzeichnis befindet sich die Datei DozentModel.php. Gemäß der MVC Definition befinden sich alle Datenbankabfragen in dem Ordnerpfad application/model. Als nächsten Schritt soll der Programmfluss, angefangen bei der Abfrage der Daten aus der Datenbank näher betrachtet werden. In **Abbildung 4** wird die Funktion „getDozentProfil“ beschrieben. Aus der Abbildung ist ersichtlich, dass die Methode mit dem Parameter „\$user“ aufgerufen wird. Der Parameter beinhaltet den aktuell eingeloggten Nutzernamen.

```

/*
 * START SPRINT 04
 * @author: Damian Wysocki
 * User Story (Nr.: 390) Als Dozent möchte ich mir mein Profil anzeigen und bearbeiten können
 * Task: 390/01 Beschreibung: SQL Queries erstellen
 * Zeitaufwand (in Stunden): 3
 * START SPRINT 04
 */
public static function getDozentProfil($user) {
    $database = DatabaseFactory::getFactory ()->getConnection ();

    //

    $sql = "SELECT ad.straßenname AS Straße,
                ad.hausnummer as Hausnummer,
                ad.plz AS PLZ,
                ad.stadt AS Stadt,
                ad.land AS Land,
                e.email_name AS Email,
                w.nutzer_name As Nutzername,
                w.vorname as Vorname,
                w.nachname as Nachname,
                w.telefonnummer as Telefonnummer
        FROM Nutzer w, EMail e, Adresse ad
        WHERE w.nutzer_name = :user
        AND ad.nutzer_name = :user
        AND e.nutzer_name = :user;";

    $query = $database->prepare ( $sql );

    try {
        $query->execute ( array (
            ':user' => $user
        ) );
    } catch ( PDOException $fehler ) {
        Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' . $fehler );
    }

    // Test
    //print_r($query->fetchAll());

    return $query->fetch();
}

```

Abbildung 77: Model zur Abfrage der Profildaten

Weiterhin werden die einzelnen Attribute eines Dozenten mit einem SELECT-Statement aus der Datenbank geholt, an den Controller übergeben und letztendlich in der View angezeigt. Die Verbindung wird über PDO mit der Funktion „**getConnection**“ aus der DatabaseFactory-Klasse realisiert. Der Controller beinhaltet die Funktion „**meinProfil**“ zur Steuerung der Aus- und Eingabe. Diese ist in **Abbildung 5** dargestellt. Hier wird der Array „**profil**“ mit der bereits erwähnten Funktion „**getDozentProfil**“ aus dem Model befüllt und an die View weitergeleitet. Der entsprechende Nutzernamen wird ebenfalls von der Session bereitgestellt, die beim Login alle relevanten Nutzerdaten speichert.

```
// Funktion um das Profil anzuzeigen
public function meinProfil() {
    $this->View->render ( 'dozent/meinProfil', array (
        ...
        'profil' => DozentModel::getDozentProfil(Session::get('user_name'))
    ) );
}
```

Abbildung 78: Controller zur Steuerung der View

Die eingangs erwähnte Datei „**meinProfilView.php**“ beinhaltet den Quellcode, der für die Anzeige in der Browserseite zuständig ist. Der Aufruf findet ebenfalls im Controller statt. Nachfolgend soll ein Ausschnitt des Quellcodes für die Ausgabe dargestellt werden. In **Abbildung 5** sieht man, dass zunächst der übergegebene Array „profil“ überprüft wird. Sollte der Array leer sein wird eine Fehlermeldung ausgegeben. Andernfalls werden die einzelnen Attribute mit der echo Funktion von PHP ausgegeben.

```

<article>
    <h1>Mein Profil</h1>
    <?php if ($this->profil) { ?>

        <form>
            <table>
                <?php // Nutzernname?>
                <tr>
                    <td><label>Nutzername</label></td>
                    <td><label><?php echo htmlentities($this->profil->Nutzername) ?></label></td>
                </tr>
                <?php // Vorname?>
                <tr>
                    <td><label>Vorname</label></td>
                    <td><label><?php echo htmlentities($this->profil->Vorname) ?></label></td>
                </tr>
                <?php // Nachname?>
                <tr>
                    <td><label>Nachname</label></td>
                    <td><input type="text" name="Nachname"
                        value=<?php echo htmlentities($this->profil->Nachname) ?>" required></td>
                    <td><input class="button" type="submit" name="nn" value="Ändern"
                        formaction="index.php?url=dozent/updateDozent" formmethod="post">
                </tr>

                <?php // Straße?>
                <tr>
                    <td><label>Straße</label></td>
                    <td><input type="text" name="Straße"
                        value=<?php echo htmlentities($this->profil->Straße) ?>" required></td>
                    <td><input class="button" type="submit" name="str" value="Ändern"
                        formaction="index.php?url=dozent/updateDozent" formmethod="post">
                </tr>
                <?php // Hausnummer?>
                <tr>
                    <td><label>Hausnummer</label></td>
                    <td><input type="text" min="1" name="Hausnummer"
                        value=<?php echo htmlentities($this->profil->Hausnummer) ?>" required></td>
                    <td><input class="button" type="submit" name="hn" value="Ändern"
                        formaction="index.php?url=dozent/updateDozent" formmethod="post">
                </tr>
            </table>
        </form>
    <?php } ?>

```

Abbildung 79: View Quellcode der Anzeige

Die einzelnen Werte werden als Tabelle gespeichert. Das Skript enthält zwei Neuerungen aus HTML5. Diese kann man in den html tags „input type sumbit“ erkennen. Es handelt sich um „formaction“ und „formmethod“. Hier lässt sich jedem einzelnen Attribut ein Action-Handling hinterlegen, sodass jedes einzelne Attribut unabhängig von den anderen Feldern geändert werden kann. Die hinterlegte URL ruft bei Klicken des Buttons die Methode „updateDozent“ aus dem Model auf. In **der Abbildung** ist der Quellcode dieser Methode dargestellt.

```

public function updateDozent() {

    $model = new DozentModel();
    // Nachname
    if (isset($_POST ['nn'])) {
    ...

    $wert = $_POST['Nachname'];

    $model-> updateDataDozentNN($wert, Session::get('user_name'));

    // Variable wieder als nicht gesetzt deklarieren
    unset($_POST['nn']);

    //print_r($_POST['Nachname']);
    }

    ...
    // Straße
    if (isset($_POST ['str'])) {
    ...

    $wert = $_POST['Straße'];

    $model-> updateDataDozentSTR($wert, Session::get('user_name'));

    // Variable wieder als nicht gesetzt deklarieren
    unset($_POST['str']);

    //print_r($_POST['Straße']);
    }
}

```

Abbildung 80: Controller Aufruf bei Ändern eines Attributs

Der Controller überprüft zunächst ob ein Feld geändert worden ist und ruft die entsprechende Funktion aus dem Model auf, die für die Änderung des Attributs zuständig ist. Der Auszug zeigt ein Beispiel für das Attribut „Straße“. Hier wird dynamisch über die Methode „updateDataDozentSTR“ die Abfrage für die Änderung aus dem Model aufgerufen. Die Funktion und das entsprechende SQL Statement hat die folgende Struktur kann in **Abbildung 8** abgelesen werden. Die Funktion nimmt die Parameter „\$wert“ und „\$user“ entgegen. Hier werden zum einen der Wert des Attributs und zum anderen der aktuell eingeloggte Nutzernamen an die Funktion übergeben. Die Werte werden als Platzhalter gemäß Dokumentation für PDO übergeben und ausgeführt.

```
// Straße
public static function updateDataDozentSTR($wert, $user) {
    $database = DatabaseFactory::getFactory ()->getConnection ();

    // print_r($wert, $user);

    // Daten in Datenbank updaten
    $sql = "UPDATE Adresse
            SET straßenname= :wert
            WHERE nutzer_name= :user;";

    $query = $database->prepare ( $sql );

    try {
        $query->execute ( array (
            ':user' => $user,
            ':wert' => $wert
        ) );
        Session::add ( 'response_positive', 'Straßennamen erfolgreich geändert.' );
    } catch ( PDOException $fehler ) {
        Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' . $fehler );
    }
    //print_r($query);
}
```

Abbildung 81: Model Straße eines Dozenten ändern

Des Weiteren werden Fehler im „try“ und „catch“ Block abgefangen. Sobald ein Attribut erfolgreich abgeändert worden ist wird eine positive Meldung angezeigt. Dies ist in der **Abbildung 9** dargestellt.

The screenshot shows a web interface with a navigation bar at the top containing 'Home', 'Mein Profil', 'Raumplan', and 'Stun...'. Below the navigation bar, a green success message box displays the text 'Straßennamen erfolgreich geändert.'. The main content area is titled 'Mein Profil'. It contains a table with the following data:

Nutzername	dozent
Vorname	Klaus
Nachname	<input type="text" value="Kopf"/>
Straße	<input type="text" value="Milöckerweg"/>
Hausnummer	<input type="text" value="10"/>
PLZ	<input type="text" value="89073"/>
Stadt	<input type="text" value="Ulm"/>
Land	<input type="text" value="Deutschland"/>
Telefonnummer	<input type="text" value="07310547874"/>
Email	<input type="text" value="dozent@hs-ulm.de"/>

Next to each input field is a blue 'Ändern' button. At the bottom of the form is a blue 'Daten Neu Laden' button.

Abbildung 82: Positive Fehlermeldung nach Änderung

Hier kann man erkennen, dass ein grüner Balken mit der positiven Meldung „Straßennamen erfolgreich geändert“ zurückgegeben wird. Sollte dies nicht der Fall sein, wird ein roter Balken mit der entsprechenden Fehlermeldung angezeigt.

5.12 Noten für Studenten eintragen

Die Speicherung und Archivierung von Noten einzelner Studenten gehört auch zu den grundlegenden Funktionen eines Lehrveranstaltung Systems. Daher wurde folgende User Story formuliert:

„Nr.: xxx, Als Mitarbeiter möchte ich Noten für Teilnehmer eintragen können“

Hier wurde ebenfalls das Konzept des MVC Patterns übernommen. Die User Story wurde so umgesetzt, dass es zwei Möglichkeiten gibt Noten einzutragen. Zum einen die Noten aller Teilnehmer einer Vorlesung und zum anderen die Note eines einzelnen Studenten. Die folgende Abbildung zeigt diese beiden Möglichkeiten.

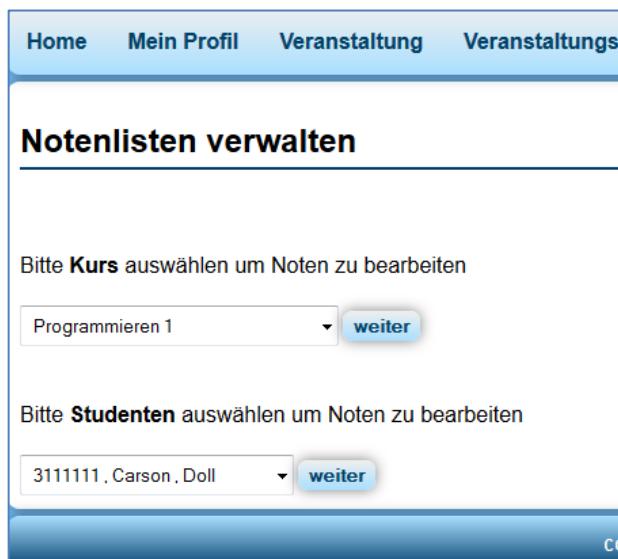


Abbildung 83: View Auswahl Kurs oder Student

Nachdem eine der folgenden Möglichkeiten ausgewählt wurde, wird die folgende Abbildung bzw. geladen.

Veranstaltungsnummer	Bezeichnung	Matrikel	Vorname	Nachname	Nutzer	Note	
1	Programmieren 1	13111111	Carson	Doll	cadoll	3.30	1.0 ▾ Note eintragen
1	Programmieren 1	51111111	Dolan	Ritch	doritc	1.70	1.0 ▾ Note eintragen
1	Programmieren 1	41111111	Logan	Luke	loluke	1.70	1.0 ▾ Note eintragen

Zurück zur Auswahl

copyright und andere informationen über diese seite

Abbildung 84: View Note für Teilnehmer ändern

Es werden alle Teilnehmer geladen, die einen Eintrag in der Notenliste Tabelle haben bzw. die bei Anlage eines Studenten in der Notenliste hinterlegt werden. Das Dropdown-Feld enthält alle möglichen Notenstufen. Der Button „Note eintragen“ ändert schließlich die Note dauerhaft in der Datenbank.

5.13 Veranstaltungstermin erstellen

5.14 Veranstaltungstermin-Übersicht mit Optionen löschen oder ändern

5.15 Veranstaltung bearbeiten

Nachdem Veranstaltungen angelegt werden können, müssen sie auch bearbeitet werden können. Dies geschieht über die Auswahl der Veranstaltung auf der Seite „?url=veranstaltung/bearbeitenSelect“.

Veranstaltungsnummer	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart	bearbeiten
1	Programmieren 1	PROG1	5	4	32	Vorlesung/Übung	bearbeiten
2	Grundlagen BWL	ABWL	5	4	32	Vorlesung/Übung	bearbeiten
3	Grundzüge Wirtschaftsinformatik	GRWF	5	4	32	Vorlesung/Übung	bearbeiten
4	Rechnungswesen	RW	5	4	32	Vorlesung/Übung	bearbeiten
5	Mathematik 1	MATH	5	4	32	Vorlesung/Übung	bearbeiten
6	Technologische Grundlagen	TEGR	5	4	16	Vorlesung/Übung	bearbeiten
7	Programmieren 2	PROG2	5	4	32	Vorlesung/Übung	bearbeiten
8	Schwerpunkt BWL	SBWL	5	4	32	Vorlesung/Übung	bearbeiten
9	Business Englisch	BUSE-NU	5	4	32	Vorlesung/Übung	bearbeiten
10	Projektmanagement	PMAN	5	4	32	Vorlesung/Übung	bearbeiten
11	Mathematik 2	MATH	5	4	32	Vorlesung/Übung	bearbeiten
12	Wirtschafts- und IT-Recht	WIPR	5	4	32	Vorlesung/Übung	bearbeiten

Die obige View wird zusammengesetzt aus einer generischen Tabelle. Dieser wird

im Controller ein Array mit allen vorhandenen Veranstaltungen übergeben, die diese darstellt. Zusätzlich wird hinter jede Zeile eine Form mit hidden values gelegt, deren Werte bei einem Klick auf den Button „bearbeiten“ an die Controller-Methode bearbeiten() übergeben wird.

```

164*      /*
165   * ruft alle vorhandenen Veranstaltungen aus der Datenbank ab und übergibt sie
166   * der view bearbeitenSelect, wo eine Veranstaltung zur Bearbeitung ausgewählt werden kann
167   */
168 public function bearbeitenSelect() {
169
170     // neues Veranstaltungmodell anlegen
171     $vModel = new VeranstaltungModel ();
172
173     // holt alle angelegten Veranstaltungen aus der Datenbank und speichert sie in einem Array
174     $veranstaltungen = $vModel->getAlleVeranstaltungen();
175
176     // übergibt die Veranstaltungen der View, in der die zu bearbeitende Veranstaltung
177     // ausgewählt werden kann
178     $this->View->render( 'veranstaltung/bearbeitenSelect', array (
179         'veranstaltungen' => $veranstaltungen
180     ) );
181 }

```



```

292 // gibt ein array zurück mit allen vorhandenen Veranstaltungen
293 public function getAlleVeranstaltungen() {
294     // Query-String, mit dem alle Veranstaltungen ausgelesen werden
295     $sq = "select Veranstaltung.veranst_ID, Veranstaltung.veranst_bezeichnung, Veranstaltung.veranst_kurztext,
296           . "Veranstaltung.credits, Veranstaltung.SWS, "
297           . "Veranstaltung.maxTeilnehmer, Veranstaltungsart.vArt_bezeichnung as Veranstaltungsart "
298           . "from Veranstaltung join Veranstaltungsart on Veranstaltung.vArt_ID = Veranstaltungsart.vArt_ID";
299           . " where Veranstaltung.veranst_ID = $vID;";
300
301     // führt die Abfrage aus und speichert das resultset in $result
302     $result = $this->abfrage($sq);
303
304     return $result;
305 }

```

Nach einem Klick auf bearbeiten wird die ID der ausgewählten Veranstaltung per POST an den Server übergeben. In der Methode bearbeiten wird ein neues Objekt vom Typ VeranstaltungModel erstellt und die Funktion getGrunddaten() ausgeführt.

```

184 // holt alle Grunddaten der Veranstaltung aus der Datenbank und übergibt sie der view "bearbeiten"
185 public function bearbeiten() {
186     // neues Veranstaltungmodell anlegen
187     $vModel = new VeranstaltungModel ();
188
189     //POST-Daten lesen
190     $vID = Request::post("veranst_ID");
191
192     if(isset($vID)) {
193
194         // holt die Veranstaltung mit vID aus der Datenbank und speichert sie in $veranstaltung
195         // $veranstaltung = $vModel->getVeranstaltung($vID);
196         $veranstaltung = $vModel->getGrunddaten($vID);
197         $this->View->render('veranstaltung/bearbeiten', array ('veranstaltung' => $veranstaltung
198     ) );
199
200     }
201 }

```

Die Methode getGrunddaten() liest Bezeichnung, Kurztext, Credits, SWS, maximale Teilnehmer und Veranstaltungsart-ID aus und liefert diese in einem Array zurück.

```

307     // holt die Grunddaten einer Veranstaltung mit $vID
308     // (Bezeichnung, Kurztext, SWS, Credits, max. Anzahl Teilnehmer, VeranstaltungsartID) aus der Datenbank
309     public function getGrunddaten($vID) {
310
311         //Query-String
312         $q = "select veranst_ID, veranst_bezeichnung, veranst_kurztext, "
313         . "credits, SWS, maxTeilnehmer, vArt_ID "
314         . "from Veranstaltung"
315         . " where veranst_ID = $vID;";
316
317         //führt die Abfrage aus und speichert sie in $result
318         $result = $this->abfrage($q);
319
320         return $result;
321     }
322

```

Die ausgelesenen Informationen werden in der View wie folgt dargestellt:

Veranstaltung bearbeiten: Grunddaten

Bezeichnung:	Grundzüge Wirtschaftsinformatik
Kurztext:	GRWF
SWS:	4
Credits (ECTS-Punkte):	5
maximale Anzahl Teilnehmer:	32

weiter

Die Daten werden in die Textfelder eingetragen bzw. bei den select-Elementen vorausgewählt. Zusätzlich wird die Veranstaltungs-ID als hidden value in der HTML-Seite gespeichert und beim Abschicken mit übergeben.

```

33 <?php $veranst = $this->veranstaltung[0]; ?>
34
35     <form action="index.php?url=veranstaltung/bearbeitenVeranstaltungsart" method="post">
36
37         <input name="veranst_ID" value=<?php echo "\"$veranst->veranst_ID\"?> type="hidden">
38         <input name="vArt_ID" value=<?php echo "\"$veranst->vArt_ID\"?> type="hidden">
39
40         <table id="veranstaltung-bearbeiten">
41             <tr><td>Bezeichnung: </td> <td><input type="text" name="veranstaltung_bezeichnung"
42                 value=<?php echo "\"$veranst->veranst_bezeichnung\"?> size="36" maxlength="100" required/></td></tr>
43             <tr><td>Kurztext: </td> <td><input type="text" name="veranstaltung_kurztext"
44                 value=<?php echo "\"$veranst->veranst_kurztext\"?> required /></td></tr>
45             <tr><td>SWS: </td> <td>
46
47                 <select name="veranstaltung_sws" style="width: 3em">
48                     <?php
49                         //befüllt den selection-Abschnitt mit den Werten von 1 bis 10
50                         //wählt den in der DABA stehenden Wert aus ("selected");
51                         for ($i=1; $i<=10; $i++) {
52                             echo "\n\t<option value=\"" . $i . "\"";
53
54                             if($i == $veranst->SWS) {
55                                 echo " selected";
56                             }
57
58                             echo ">" . $i . "</option>";
59                         }
60                     ?>
61                 </select>
62             </td></tr>

```

Nachdem die Grunddaten bearbeitet worden sind, wird im Controller die Methode bearbeitenVeranstaltungsart() aufgerufen.

```

204* /*
205 * wird von "bearbeiten" aufgerufen, bekommt die Grunddaten (Bezeichnung, Kurztext, SWS,
206 * Credits, Maximale Anzahl Teilnehmer) übergeben, die am Ende in der Datenbank geupdated werden sollen.
207 *
208 * Holt die Veranstaltungsarten aus der Datenbank und zeigt diese im nächsten Fenster des
209 * Bearbeitungs-Prozesses an.
210 *
211 */
212@ public function bearbeitenVeranstaltungsart() {
213
214     // Array mit "Grunddaten"
215     $grunddaten = array (
216         'vID'          => Request::post("veranst_ID"),
217         'vBezeichnung' => Request::post("veranstaltung_bezeichnung"),
218         'vKurztext'    => Request::post("veranstaltung_kurztext"),
219         'vSWS'          => Request::post("veranstaltung_sws"),
220         'vCredits'     => Request::post("veranstaltung_credits"),
221         'vMaxTeilnehmer'=> Request::post("veranstaltung_max_Teilnehmer"),
222         'vArt_ID'       => Request::post("vArt_ID")
223     );
224
225     // neues Veranstaltungmodell anlegen
226     $vModel = new VeranstaltungModel ();
227
228     //Veranstaltungs-ID
229     $vID = $grunddaten['vID'];
230
231     if(isset($vID)) {
232
233         // holt die Veranstaltung mit vID aus der Datenbank und speichert sie in $veranstaltung
234         $vArten = $vModel->getVeranstaltungsarten();
235
236         $this->View->render('veranstaltung/bearbeitenVeranstaltungsart', array ('grunddaten' => $grunddaten,
237             'vArten' => $vArten
238         ) );
239
240     }
241 }

```

Die bearbeiteten, per POST übergebenen Grunddaten werden im Array \$grunddaten gespeichert. Per Model-Objekt werden die Veranstaltungsarten (s.o.) ausgelesen und zusammen mit den Grunddaten der View bearbeitenVeranstaltungsart übergeben.

Veranstaltung bearbeiten: Veranstaltungsart

"Grundzüge Wirtschaftsinformatik"

Veranstaltungsart: Vorlesung/Übung

weiter

Hier werden die Veranstaltungsarten in einem select-Element angezeigt, die aktuell gespeicherte Veranstaltungsart ist dabei vorgewählt.

```

18 <h1>Veranstaltung bearbeiten: Veranstaltungsart</h1>
19 <?php
20 |
21 $grunddaten = $data['grunddaten'];
22 $vArten = $data['vArten'];
23
24 ?>
25 <h2><?php echo $grunddaten['vBezeichnung']?></h2>
26 <form action="index.php?url=veranstaltung/bearbeitenAusstattung" method="post">
27 <?php
28 foreach($grunddaten as $key => $value) {
29     echo "<input name=\"" . $key . "\" value=\"" . $value . "\" type=\"hidden\">\n";
30 }
31 ?>
32
33<table id="veranstaltungsart-bearbeiten">
34 <tr><td>Veranstaltungsart: </td><td>
35 <select class="input" name="veranstaltung_veranstaltungsart" style=" width: 32em">
36
37 <?php
38 foreach($this->vArten as $key => $value) {
39     echo "\n\t<option value=\"" . $value->vArt_ID . "\"";
40
41     if($value->vArt_ID == $grunddaten['vArt_ID']) {
42         echo " selected";
43     }
44     echo ">";
45     echo utf8_encode($value->vArt_bezeichnung) . "</option>";
46 }
47 ?>
48     </select>
49 </td></tr>
50
51     <tr><td><input class="button" type="submit" value="weiter" /></td><td>&ampnbsp</td></tr>
52 </table>
```

Außerdem werden die Grunddaten als Hidden Values in die HTML-Datei geschrieben, damit diese bis zum Ende des Bearbeitungs-Prozess' weitergegeben werden können.

Mit einem Klick auf Weiter wird die Methode bearbeitenAusstattung() im Controller aufgerufen.

```

253 public function bearbeitenAusstattung() {
254
255     // POST-Daten lesen
256     // Array mit "Grunddaten"
257     $grunddaten = array (
258         'vID'      => Request::post("vID"),
259         'vBezeichnung' => Request::post("vBezeichnung"),
260         'vKurztext' => Request::post("vKurztext"),
261         'vWS'       => Request::post("vWS"),
262         'vCredits'  => Request::post("vCredits"),
263         'vMaxTeilnehmer'=> Request::post("vMaxTeilnehmer")
264         // 'Veranstaltungsart' => Request::post("Veranstaltungsart")
265     );
266
267     // (neu gewählte) Veranstaltungsart als ID
268     $vArtID = Request::post("veranstaltung_veranstaltungsart");
269
270     // neues Veranstaltungmodell anlegen
271     $vModel = new VeranstaltungModel ();
272
273     // $ausstattung = $vModel->getAusstattung();
274     $ausstattung = $vModel->getAusstattungVonVeranstaltung(Request::post("vID"));
275
276
277     $this->View->render('veranstaltung/bearbeitenAusstattung', array ('grunddaten' => $grunddaten,
278         'vArtID' => $vArtID,
279         'ausstattung' => $ausstattung
280     ) );
281 }
```

In der Methode werden die eingetragenen Ausstattungsdetails ausgelesen und zusammen mit den Grunddaten weiter gereicht.

Veranstaltung bearbeiten: Ausstattung

"Grundzüge Wirtschaftsinformatik"

16	Stuhl
0	Computer
1	Beamer
1	Projektor
1	Overheadprojektor
1	Praesentations-Computer
1	HDMI-Anschluss
1	Whiteboard
0	Kreidetafel
0	Telefon

weiter

Abbildung 85: Veranstaltung bearbeiten – View

Auch hier werden die zuvor geänderten Werte als hidden values gespeichert.

Zudem werden die Ausstattungsmerkmale eingetragen. Der zugrunde liegende

Code:

```

17 <article>
18 <h1>Veranstaltung bearbeiten: Ausstattung</h1>
19 <?php
20
21 $grunddaten = $data['grunddaten'];
22 $vArtID = $data['vArtID'];
23 $ausstattung = $data['ausstattung'];
24 ?>
25     <?php echo $grunddaten['vBezeichnung']?></h2>
26     <form action="index.php?url=veranstaltung/bearbeitenEintragen" method="post">
27     <?php
28     //die bereits übergebenen Werte aus vorigen Fenstern werden hier als hidden values eingetragen,
29     //damit diese weitergegeben werden können
30     foreach($grunddaten as $key => $value) {
31         echo "<input name=\"\" . $key . \"\" value=\"\" . $value . \"\" type=\"hidden\">\n\t";
32     }
33     echo "<input name=\"vArtID\" value=\"\" . $vArtID . \"\" type=\"hidden\">\n";
34     ?>
35     <table id="veranstaltungausstattung-bearbeiten">
36     <?php
37     // erzeugt eine Tabelle mit allen Ausstattungsmerkmalen
38     foreach($this->ausstattung as $key => $value) {
39         echo "\n<tr>";
40         //ausstattung[] gruppiert die so benannten Eingabefelder;
41         //in PHP wird dies zu einem Sub-Array zusammengefasst
42         // TODO hidden value für id?
43         echo "\n\t<td><input type=\"text\" name=\"veranstaltung_ausstattung[]\" "
44         . "value=\"\" . $value->anzahl;
45         echo "\" size=\"1\" /> </td>";
46         echo "\n\t<td> " . utf8_encode($value->ausstattung_bezeichnung);
47         echo "</td>";
48         echo " \n</tr>";
49     }
50     ?>
51     <tr><td><input class="button" type="submit" value="weiter" /></td><td>&ampnbsp</td></tr>
52 </table>
```

Ein Klick auf weiter übträgt alle (geänderten) Daten an die Methode bearbeitenEintragen() im VeranstaltungController. Hier werden die Grunddaten wieder zusammengefasst, Veranstaltungsart-ID und Ausstattung in Variablen

gespeichert. Zuerst werden die neuen Zahlen für die Ausstattung eingetragen. Danach werden die neu eingetragenen Werte ausgelesen, damit diese auf der nächsten Seite ausgegeben werden können. Im Anschluss wird updateVeranstaltung() mit den drei Arrays Grunddaten, Veranstaltungsart-ID und Ausstattung aufgerufen und gleich danach mit getVeranstaltung() ausgelesen. Wenn alles funktioniert hat, wird der Session ein positive_response geschrieben und anschließend die View bearbeitet aufgerufen.

```

289 |     public function bearbeitenEintragen() {
290 |
291 |         // Array mit "Grunddaten", ohne Veranstaltungsart
292 |         $grunddaten = array (
293 |             'vID'          => Request::post("vID"),
294 |             'vBezeichnung' => Request::post("vBezeichnung"),
295 |             'vKurztext'    => Request::post("vKurztext"),
296 |             'vSWS'         => Request::post("vSWS"),
297 |             'vCredits'    => Request::post("vCredits"),
298 |             'vMaxTeilnehmer'=> Request::post("vMaxTeilnehmer")
299 |         );
300 |         // (neu gewählte) Veranstaltungsart als ID
301 |         $vArtID = Request::post("vArtID");
302 |         // Array mit benötigter Ausstattung
303 |         $ausstattung = Request::post("veranstaltung_ausstattung");
304 |         // neues Veranstaltungmodel anlegen
305 |         $vModel = new VeranstaltungModel ();
306 |         // trägt Ausstattung ein
307 |         if($vModel->ausstattungEintragen(Request::post("vID"), $ausstattung)) {
308 |             $ausstattungNeu = $vModel->getAusstattungVonVeranstaltung(Request::post("vID"));
309 |
310 |             // alte Veranstaltung, für Vorher-Nachher
311 |             $alteVeranstaltung = $vModel->getVeranstaltung(Request::post("vID"));
312 |
313 |             if($vModel->updateVeranstaltung(array('grunddaten' => $grunddaten,
314 |                                                 'vArtID' => $vArtID,
315 |                                                 'ausstattung' => $ausstattung ))) {
316 |                 // neue Veranstaltung, für Vorher-Nachher
317 |                 $neueVeranstaltung = $vModel->getVeranstaltung(Request::post("vID"));
318 |
319 |                 Session::add ( 'response_positive', 'Veranstaltung erfolgreich bearbeitet.' );
320 |
321 |                 $this->View->render('veranstaltung/bearbeitet', array (
322 |                     'alteVeranstaltung' => $alteVeranstaltung,
323 |                     'neueVeranstaltung' => $neueVeranstaltung,
324 |                     'ausstattung' => $ausstattungNeu
325 |                 ) );
326 |             } else {
327 |                 Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
328 |             }
329 |         } else {
330 |             Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
331 |         }
332 |
333 |
334 }
```

```

324|     public function updateVeranstaltung($data) {
325
326         $grunddaten = $data['grunddaten'];
327         $vID        = $grunddaten['vID'];
328         $vBezeichnung = $grunddaten['vBezeichnung'];
329         $vKurztext   = $grunddaten['vKurztext'];
330         $vSWS        = $grunddaten['vSWS'];
331         $vCredits    = $grunddaten['vCredits'];
332         $vMaxTeilnehmer = $grunddaten['vMaxTeilnehmer'];
333         $vArtID      = $data['vArtID'];
334         $ausstattung = $data['ausstattung'];
335         $database = new DatabaseFactoryMysql ();
336
337         $updateString = "UPDATE Veranstaltung SET veranst_bezeichnung = '$vBezeichnung', "
338             . "veranst_kurztext = '$vKurztext', credits = $vCredits, SWS = $vSWS, "
339             . "maxTeilnehmer = $vMaxTeilnehmer, vArt_ID = $vArtID "
340             . "where veranst_ID = $vID;";
341
342         if($database->insert($updateString)) {
343
344             // löscht Einträge in der Datenbank Veranstaltung_erfordert_Ausstattung
345             if($this->loescheAusstattung($vID)) {
346                 // trägt die neuen Ausstattungsmerkmale ein
347                 return $this->ausstattungEintragen($vID, $ausstattung);
348             }
349         }
350     }
351

```

Es werden in einer großen Update-Query die Veranstaltung geändert.

Veranstaltung erfolgreich bearbeitet.

voher:						
Veranstaltungsnummer	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart
3	Grundzüge Wirtschaftsinformatik GRWF		5	4	32	Vorlesung/Übung

nachher:						
Veranstaltungsnummer	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart
3	Grundzüge Wirtschaftsinformatik GRWF		5	4	16	Seminar/Vorlesung

5.16 Veranstaltung als Wahlpflichtfach

Manchmal sollen Veranstaltungen als Wahlfach für bestimmte Studiengänge festgelegt werden. Dies geschieht im Menüpunkt „als Wahlfach“.

Die Auswahl ist die gleiche wie bei Veranstaltung bearbeiten: Es wird ein Array der vorhandenen Veranstaltungen die View übergeben, die mithilfe einer generischen Tabelle eine Übersicht samt HTML-Form mit hidden values erstellt.

Wahlfach: Veranstaltung auswählen							
Wählen Sie die Veranstaltung aus, die Sie einem Studiengang als Wahlfach hinzufügen möchten:							
Veranstaltungsnr.	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart	
1	Programmieren 1	PROG1	5	4	32	Vorlesung/Übung	bearbeiten
2	Grundlagen BWL	ABWL	5	4	32	Vorlesung/Übung	bearbeiten
3	Grundzüge Wirtschaftsinformatik	GRWF	5	4	16	Seminar/Vorlesung	bearbeiten
4	Rechnungswesen	RW	5	4	32	Vorlesung/Übung	bearbeiten
5	Mathematik 1	MATH	5	4	32	Vorlesung/Übung	bearbeiten
6	Technologische Grundlagen	TEGR	5	4	16	Vorlesung/Übung	bearbeiten
7	Programmieren 2	PROG2	5	4	32	Vorlesung/Übung	bearbeiten

Abbildung 86: Veranstaltung als Wahlfach – View – wahlfachVeranstaltungSelect

Nachdem der Benutzer auf bearbeiten geklickt hat, wird die Methode `bearbeitenWahlfach()` im Controller aufgerufen; diese lädt die gewählte Veranstaltung und die Studiengänge, denen sie zugeordnet werden kann.

```

364     // zeigt alle Studiengänge an, zu der die Veranstaltung hinzugefügt werden kann
365     public function bearbeitenWahlfach() {
366         // neues Veranstaltungmodell anlegen
367         $vModel = new VeranstaltungModel();
368
369         //POST-Daten lesen
370         $vID = Request::post("veranst_ID");
371
372         if(isset($vID)) {
373
374             // holt die Veranstaltung mit vID aus der Datenbank und speichert sie in $veranstaltung
375             $veranstaltung = $vModel->getVeranstaltung($vID);
376
377             //enthält alle Studiengänge, für die die Veranstaltung noch nicht hinzugefügt wurde
378             $studgaenge = $vModel->getStudiengaengeWahlfach($vID);
379
380             $this->View->render('veranstaltung/wahlfachbearbeiten',
381                             array ('veranstaltung' => $veranstaltung,
382                                    'studiengaenge' => $studgaenge
383                             ) );
384
385         }
386     }

```

Abbildung 87: Veranstaltung als Wahlfach – Controller – wahlfachVeranstaltungSelect

Mit der Methode `getStudiengaengeWahlfach()` werden alle Studiengänge aus der Datenbank abgerufen, die dem Studiengang hinzugefügt werden können.

```

589     public function getStudiengaengeWahlfach($vID) {
590
591         // query-string
592         $q = "select stg_ID, stg_bezeichnung, stgTyp_kuerzel from Studiengang "
593             . "join Studiengangtyp on Studiengang.stgTyp_ID = Studiengangtyp.stgTyp_ID";
594
595         // erzeugt ein Resultset, benutzt dazu die Methode abfrage($q)
596         $result = $this->abfrage( $q );
597
598         return $result;
599     }
600

```

Mit dem Select kann der Studiengang gewählt werden, dem die Veranstaltung als Wahlfach zugeordnet werden soll.

Wahlfach: Veranstaltung auswählen

Wählen Sie den Studiengang aus, dem Sie die Veranstaltung als Wahlfach zuordnen möchten:

Veranstaltungsnummer	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart
2	Grundlagen BWL ABWL	5	4	32		Vorlesung/Übung

Studiengang:

1 - Digital Media BA ▾ ab Fachsemester: 1 ▾

weiter

Ein Klick auf weiter überträgt den Inhalt des Selects an den Server. Die Methode wahlfachEintragen() wird im Controller aufgerufen.

```

392e public function wahlfachEintragen() {
393     // neues Veranstaltungmodell anlegen
394     $vModel = new VeranstaltungModel ();
395
396     //POST-Daten lesen
397     $vID = Request::post("veranst_ID");
398     $stg_ID = Request::post("studiengang");
399     $semester = Request::post("semester");
400
401     if(isset($vID) && isset($stg_ID)) {
402
403         if($vModel->setWahlfach($stg_ID, $vID, $semester)) {
404
405             // holt die Veranstaltung mit vID aus der Datenbank und speichert sie in $veranstaltung
406             // $veranstaltung = $vModel->getVeranstaltung($vID);
407             //$veranstaltung = $vModel->getGrunddaten($vID);
408             $veranstaltung = $vModel->getVeranstaltung($vID);
409
410             //enthält alle Studiengänge, für die die Veranstaltung noch nicht hinzugefügt wurde
411             $studgang = $vModel->getStudiengang($stg_ID);
412
413             Session::add ( 'response_positive', 'Wahlfach erfolgreich eingetragen.' );
414
415             $this->View->render('veranstaltung/wahlfacheingetragen',
416                 array ('veranstaltung' => $veranstaltung,
417                         'studiengang' => $studgang,
418                         'semester' => $semester
419                     ) );
420         } else {
421             Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
422         }
423     }
424 }
```

Wird die Methode setWahlfach() erfolgreich ausgeführt, wird der Session ein repsonse_positive übergeben.

```

621e public function setWahlfach($studiengang, $vID, $semester) {
622     // Datenbankverbindung
623     $database = new DatabaseFactoryMysql ();
624
625     $insertString = "INSERT INTO Studiengang_hat_Wahlfach (stg_ID, veranst_ID, ab_Semester) "
626     . " VALUES ($studiengang, $vID, $semester);";
627
628     return $database->insert ( $insertString );
629 }
```

In der View „wahlfacheingetragen“ wird dann die Veranstaltung, der Studiengang und das Semester angezeigt.

Wahlfach erfolgreich eingetragen.						
Veranstaltungsnummer	Bezeichnung	Kurztext	Credits	SWS	maximale Anzahl Teilnehmer	Veranstaltungsart
2	Grundlagen BWL ABWL	5	4	32		Vorlesung/Übung
Studiengangnummer	Bezeichnung	Typ				
1	Digital Media	BA				
ab Semester : 1						
zurück						

5.17 Veranstaltungs-ID erstellen

Es wird eine ID für Veranstaltungen erstellt, die dem Muster FFSSVVVV entsprechen, wobei FF für Fakultäts-ID, SS für Studiengang-ID und VVVV für die Veranstaltung in Zehnerschritten steht. Der Methode wird eine Studiengangs-ID übergeben. Diese wird genutzt, um die Fakultäts-ID zu ermitteln. Sollte die Fakultäts-ID kleiner 10 sein, wird eine Null vorangestellt. Sollte die Studiengangs-ID kleiner 10 sein, wird ihr ebenfalls eine Null vorangestellt. Als nächstes wird die bisher höchste Veranstaltungs-ID gesucht, die das selbe FFSS-Muster haben. Der Code wird in der fertigen Software nicht benutzt.

```

470 public function erstelleVID($stg_ID) {
471
472     $vID = "0";
473     if($stg_ID > 0) {
474         //Query-String
475         $q = "select fak_ID from Studiengang where stg_ID = $stg_ID;";
476
477         //Query ausführen
478         $result = $this->abfrage($q);
479
480         //Enthält die Fakultäts-ID
481         $fak_ID = $result[0]->fak_ID;
482
483         //vID soll aussehen: FFSSVVVV. Dafür ist es u.U. nötig, der Fakultäts-ID eine führende Null zu geben
484         if($fak_ID < 10) {
485             $fak_ID = str_pad($fak_ID, 2, '0', STR_PAD_LEFT);
486         }
487
488         //vID soll aussehen: FFSSVVVV. Dafür ist es u.U. nötig, der Fakultäts-ID eine führende Null zu geben
489         if($stg_ID < 10) {
490             $stg_ID = str_pad($stg_ID, 2, '0', STR_PAD_LEFT);
491         }
492         //erste Hälfte der erstellten $vID zusammensetzen
493         $vID = $fak_ID . $stg_ID;
494     } else {
495         //wenn die Veranstaltung keinem Studiengang zugeordnet werden soll, nimm 0000 als führende Ziffern
496         $vID = "0000";
497     }
498     //Query-String
499     //gesucht wird die größte vID, die bereits der Fakultät/dem Studiengang zugeordnet ist
500     //Anm.: QUOTING wichtig: '_expr_','%expr%')
501     $q = "select max(veranst_ID) as vID from Veranstaltung where veranst_ID like '" . ($vID . "___") . "'"; 
502
503     //Query ausführen
504     $result = $this->abfrage($q);
505
506     //Sollte noch keine Veranstaltungs-ID für den Studiengang vorhanden sein, nimm 0000
507     $max_vID = "0";
508     if(empty($result[0]->vID)) {
509         $max_vID = "0000";
510     } else {
511         $max_vID = $result[0]->vID;
512         //die ersten vier Ziffern (FFSS) entfernen
513         $max_vID = substr($max_vID, -4);
514     }

```

Die höchste vorhandene Veranstaltungs-ID wird hergenommen, um die ID für die

Rückgabe zu ermitteln. Dafür wird sie auf den nächsten 10er-Schritt erhöht. Sollte die Veranstaltungs-ID weniger als 4 Ziffern lang sein, werden links noch Nullen vorangestellt.

```

515 |     //entferne führende Nullen
516 |     $count = 0;
517 |
518 |     //zählt nur so lange, wie Nullen links stehen; bei erster Ziffer ungleich 0 wird abgebrochen
519 |     for($i=1; $i<3; $i++) {
520 |         if($max_vID[$i] != "0") {
521 |             $i=3;
522 |         } else {
523 |             $count++;
524 |         }
525     }
526 |
527 |     //liefert Substring ab $count (exklusiv)
528 |     $max_vID = substr($max_vID, $count);
529 |     //    // größte vorhandene Veranstaltungs-ID auf nach unten 10 runden und 10 addieren
530 |     $max_vID = $max_vID - ($max_vID % 10) + 10;
531 |
532 |     //    // füllt den String VVV links mit Nullen auf
533 |     $max_vID = str_pad($max_vID, 4, '0', STR_PAD_LEFT);
534 |     //    //setze die Veranstaltungs-ID zusammen
535 |     $vID = $vID . $max_vID;
536 |
537 |     return $vID;
538 }
```

5.18 Veranstaltung erweitern

Unter der Erweiterung einer Veranstaltung versteht man bei dieser Funktionalität, dass ein Student oder Dozent der Veranstaltung zugeordnet werden soll. Hierzu sind über das Projekt hinweg zwei User Storys zuzuteilen. „Als Mitarbeiter möchte ich einen Lehrverantwortlichen / Dozent / Tutor zu einer Veranstaltung hinzufügen können. (Nr. 30)“ und „Als Mitarbeiter möchte ich einer Veranstaltung einen Dozenten zuordnen können. (Nr. 560)“. Diese Funktionalität umfasst des Weiteren, dass ein Student zu einer Veranstaltung hinzugefügt werden kann. Deswegen muss zunächst ausgewählt werden, ob die Erweiterung einen Studenten oder einen Dozenten betrifft.



Abbildung 88: Darstellung Auswahl der Erweiterung

Sobald eine Auswahl getroffen ist und auf „weiter“ geklickt wurde, wird die typSelected-Funktion im Contoller aufgerufen. Diese erzeugt die

umPersonErweitern-View, übergibt alle Veranstaltungen als Array und übergibt je nach Wahl der Personenart ein Array mit allen Studenten (getStudent-Funktion des veranstaltungErweiternModels) oder allen Dozenten (getDozent-Funktion des veranstaltungErweiternModels).

```
public function typSelected() {
    if (isset($_POST['typ'])) {
        $typ = $_POST['typ'];
        switch ($typ) {
            case "dozent":
                //umPersonErweitern-View wird gezeigt und die arrays veranstaltung_list und user_list übergeben
                $this->View->render('veranstaltungErweitern/umPersonErweitern', array(
                    'veranstaltung_list' => veranstaltungErweiternModel::getVeranstaltung(),
                    'user_list' => veranstaltungErweiternModel::getDozent()
                ));
                break;
            case "student":
                //umPersonErweitern-View wird gezeigt und die arrays veranstaltung_list und user_list übergeben
                $this->View->render('veranstaltungErweitern/umPersonErweitern', array(
                    'veranstaltung_list' => veranstaltungErweiternModel::getVeranstaltung(),
                    'user_list' => veranstaltungErweiternModel::getStudent()
                ));
                break;
            default:
                Session::add('response_negative', 'Es ist ein Fehler aufgetreten.');
        }
    } else {
        Session::add('response_warning', 'Es ist kein Personentyp ausgewählt.');
        $this->View->render('veranstaltungErweitern/veranstaltungErweitert');
    }
}
```

Abbildung 89: typSelected aus veranstaltungErweiternController

Die aufgerufene umPersonErweiternView ist wie folgt implementiert. Zuerst werden die Überschrift und der Erklärungssatz je nach Auswahl angepasst. Dies ist folgender Abbildung zu sehen.

```
<?php
$typ = $_POST['typ'];
if($typ == "dozent"){
    echo '<h1>Veranstaltung - Dozent</h1>';
    echo '<p>Hier können Sie einen Dozenten einer Veranstaltung zuweisen.</p>';
    echo "<input type='hidden' name='personentyp' value='dozent'>";
} elseif ($typ == "student") {
    echo '<h1>Veranstaltung - Student</h1>';
    echo '<p>Hier können Sie einen Studenten zu den Teilnehmern einer Veranstaltung hinzufügen.</p>';
    echo "<input type='hidden' name='personentyp' value='student'>";
}
?>
```

Abbildung 90: umPersonErweiternView Teil 1

Dann werden alle Veranstaltungen, die mit einem Array vom Controller übergeben wurden, als Dropdown-Menü dargestellt. Dies ist für beide Varianten gleich.

```
<th>Veranstaltung </th><th><select name="veranstaltung">
<?php
if ($this->veranstaltung_list) {
    foreach ( $this->veranstaltung_list as $key => $value ) {
        echo '<option>';
        echo htmlentities ( $value->veranst_ID );
        echo ",";
        echo htmlentities ( $value->veranst_bezeichnung );
        echo '</option>';
    }
} else {
    Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
}
?>
</select></th>
```

Abbildung 91: umPersonErweiternView Teil 2

In Teil 3 der umPersonErweiternView wird dann die übergebene user_list verarbeitet. Je nachdem was im ersten Schritt gewählt wurde, ist der Inhalt dieses Arrays ein anderer. Auch hier wird ein Dropdown-Menü erstellt aus dem später ausgewählt werden kann.

```
<?php
$typ = $_POST['typ'];
if($typ == "dozent"){
    echo '<th>Dozent </th>';
} else if ($typ == "student") {
    echo '<th>Student </th>';
}

echo '<th><select name="user">';
if ($this->user_list) {
    foreach ( $this->user_list as $key => $value ) {
        echo '<option>';
        echo htmlentities ( $value->nutzer_name );
        echo ",";
        echo htmlentities ( $value->rolle_bezeichnung );
        echo '</option>';
    }
} else {
    Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
}
echo '</select></th>';
?>
```

Abbildung 92: umPersonErweitern Teil 3

Die Darstellung der gerade erklärten View sieht wie folgt aus:

Veranstaltung - Dozent	ung - Student
Hier können Sie einen Dozenten einer Veranstaltung zuweisen.	einen Studenten zu den Teilnehmern einer Veranstaltung hinzufügen.
<input style="width: 150px; height: 20px; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px; margin-right: 10px;" type="text" value="1.Programmieren 1"/> Dozent <input style="width: 150px; height: 20px; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px;" type="text" value="clstei, Dozent"/>	<input style="width: 150px; height: 20px; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px; margin-right: 10px;" type="text" value="1.Programmieren 1"/> Student <input style="width: 150px; height: 20px; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px;" type="text" value="ahali, Student"/>
Zurück Veranstaltung erweitern	Zurück Veranstaltung erweitern

Abbildung 93: umPersonErweiternView

Ist dann die gewünschte Auswahl getroffen, also Veranstaltung und Person ausgewählt, kann die Veranstaltung um diese Person erweitert werden. Dabei wird die selected-Methode im Controller aufgerufen. Diese Methode macht die ausgewählten Daten zunächst einmal verwertbar. Die Veranstaltungs-ID, der User-Name und die User-Rolle sind dabei die wichtigen Informationen.

```
public function selected() {
    // Prüfung ob sowohl Veranstaltung als auch User selected sind.
    if (isset($_POST['veranstaltung']) and isset($_POST['user'])) {
        // $_POST['veranstaltung'] über gibt einen String mit der veranst_ID und der veranst_bezeichnung
        // durch explode wird dieser String getrennt, damit man mit der ID weiter arbeiten kann
        $veranst_string = $_POST['veranstaltung'];
        $veranst_array = explode(", ", $veranst_string);
        $veranst_ID = $veranst_array[0];

        // $_POST['user'] über gibt einen String mit dem user_name und der rolle_bezeichnung
        // durch explode wird dieser String getrennt, damit man mit dem user_name weiter arbeiten kann
        $user_string = $_POST['user'];
        $user_array = explode(", ", $user_string);
        $user_name = $user_array[0];
        $user_rolle = $_POST['personentyp'];
    }
}
```

Abbildung 94: selected-Funktion im veranstaltungErweiternController Teil 1

Je nach Rolle der ausgewählten Person werden dann die Erweiterungen in die Datenbank gespeichert oder aktualisiert. Beim Dozenten ist zudem wichtig, dass überprüft wird, ob er die Qualifikation besitzt die ausgewählte Veranstaltung zu leiten. Dazu nachfolgend mehr.

```
switch ($user_rolle) {
    case "student":
        // inserieren der Erweiterung
        veranstaltungErweiternModel::setErweiterungStudent($veranst_ID, $user_name);
        $this->View->render('veranstaltungErweitern/veranstaltungErweitert');
        break;
    case "dozent":
        $data = array('pruefziffer' => veranstaltungErweiternModel::pruefenDozent($veranst_ID, $user_name));
        // print_r($data);
        foreach ($data as $key => $value) {
            $this->{$key} = $value;
            // print_r($value);
        }
        foreach ($this->pruefziffer as $key => $value) {
            // echo htmlentities($value[0]);
            $zahl = $value->zah;
        }
        if ($zahl == '0') {
            Session::add('response_warning', 'Der Dozent ist nicht für die ausgewählte Veranstaltung berechtigt.');
            $this->View->render('veranstaltungErweitern/veranstaltungErweitert');
        } else if ($zahl == '1') {
            // inserieren der Erweiterung
            veranstaltungErweiternModel::setErweiterungDozent($veranst_ID, $user_name);
            $this->View->render('veranstaltungErweitern/veranstaltungErweitert');
        }
        break;
}
```

Abbildung 95: selected-Funktion im veranstaltungErweiternController Teil 2

Die Überprüfung der Qualifikation des Dozenten ist durch die pruefenDozent-Funktion im veranstaltungErweiternModel umgesetzt. Dabei wird geprüft, ob der

Dozent für die Veranstaltung einen Eintrag in der Dozent_berechtigtFuer_Veranstaltung-Tabelle hat. Falls nicht wird eine 0 zurückgegeben, falls ja eine 1. Wie in der Controller in der selected-Funktion gesehen, wird je nach Ergebnis dieser Prüfung eine Warnung ausgegeben oder die Erweiterung ausgeführt.

```
public function pruefenDozent($veranst_ID, $user_name) {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $sql = "SELECT count(dozent_nutzer_name) AS zahl
            FROM Dozent_berechtigtFuer_Veranstaltung
            WHERE dozent_nutzer_name = '$user_name' AND veranst_ID = '$veranst_ID';";
    $query = $database->prepare($sql);
    try {
        $query->execute ();
        return $query->fetchAll ();
    } catch ( PDOException $e ) {
        Session::add( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
    }
}
```

Abbildung 96: pruefenDozent aus veranstaltungErweiternModel

Doch was bedeuten die jeweiligen Erweiterungen. Im Falle eines Studenten bedeutet die dies, dass der Student zur Teilnehmerliste hinzugefügt wird.

```
public function setErweiterungStudent($veranst_ID, $user_name) {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $sql = "Insert into Student_angemeldetAn_Veranstaltung (veranst_ID, student_nutzer_name)
            Values ('$veranst_ID', '$user_name')";
    $query = $database->prepare ( $sql );
    try{
        $query->execute ();
        Session::add ( 'response_positive', 'Student wurde erfolgreich hinzugefügt.' );
    } catch ( PDOException $e ) {
        if ($e->errorInfo [1] == 1062) {
            Session::add ( 'response_warning', 'Der Student ist bereits an Veranstaltung beteiligt.' );
        } else {
            Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
        }
    }
}
```

Abbildung 97: setErweiterungStudent aus veranstaltungErweiternModel

Im Falle eines Dozenten bedeutet die Erweiterung, dass der Dozent der Veranstaltung als Verantwortlicher zugeordnet wurde. Dies geschieht über folgendes Update-Statement in der Datenbank. Dabei ist schon überprüft, ob der Dozent überhaupt qualifiziert ist, die Veranstaltung zu leiten.

```
public function setErweiterungDozent($veranst_ID, $user_name) {
    $database = DatabaseFactory::getFactory ()->getConnection ();
    $sql = "UPDATE Veranstaltung SET dozent_nutzer_name = '$user_name' WHERE veranst_ID = '$veranst_ID';";
    $query = $database->prepare ( $sql );
    try{
        $query->execute ();
        Session::add ( 'response_positive', 'Dozent wurde erfolgreich hinzugefügt.' );
    } catch ( PDOException $e ){
        Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten.' );
    }
}
```

Abbildung 98: setErweiterungDozent aus veranstaltungErweiternModel

Eine erfolgreiche Erweiterung sieht im Browser dann wie folgt aus:



Abbildung 99: Erfolgreiche Erweiterung Dozent



Abbildung 100: Erfolgreiche Erweiterung Student

5.19 . PDF-Dateien erstellen und speichern

Im Laufe des Projekts kam mit der Zeit die Anforderung auf Inhalte des Systems zu speichern. Hierzu gab es die Überlegung in welchem Format dies stattfinden soll.

Das Entwicklerteam einigte sich schnell auf das Format PDF. Hierzu war dann ein Tool nötig, welches aus den Seiteninhalten PDF-Dateien erstellen kann. Das hierfür eingeführte Tool heißt TCPDF. Dieses Werkzeug wird dazu genutzt den Inhalt, welcher als HTML-Code verfügbar ist, in eine PDF-Datei umzuwandeln und auszugeben. Als Verfasser des Tools gilt Nicola Asuni, ein asiatischer Entwickler. Über folgende Links hat man Zugriff auf die Dokumentation und den Quellcode:

- <http://www.tcpdf.org>
- <http://sourceforge.net/projects/tcpdf/files/>

Wie das Tool angewendet wird zeigen die Funktionalitäten „Notenspiegel/Zeugnis drucken“ und „Nutzerliste drucken“.

Die User Story zu der Einführung dieses Tools lautet: „Als Entwickler möchte ich ein Tool haben um PDFs ausgeben zu können. (Nr. 460)“

5.20 . Notenspiegel/Zeugnis drucken

„Als Student möchte ich mir ein Zeugnis mit allen bisher erbrachten Leistungen erstellen lassen können. (Nr. 440)“. So lautet die User Story zur Funktionalität „Notenspiegel/Zeugnis drucken“. Hierbei ist zuallererst wichtig, welcher Student eingeloggt ist. Deshalb fragt der Controller dies ab und erzeugt mit dieser Information die View.

```
public function zeugnis() {
    $current_user = Session::get('user_name');
    $this->View->render ( 'zeugnis/zeugnis', array (
        'noten_list' => NotenModel::getSpecificStudents2($current_user)
    ) );
}
```

Abbildung 101: Konstruktor zeugnisController

Übergeben wird zudem die Notenliste des Studenten, die im NotenModel abgefragt wird.

```
public function getSpecificStudents2($user_name){
    $database = DatabaseFactory::getFactory()->getConnection();
    $sql = 'SELECT n.veranst_ID, v.veranst_bezeichnung, n.note
            FROM Notenliste n
            JOIN veranstaltung v ON v.veranst_ID = n.veranst_ID
            WHERE n.student_nutzer_name = "' . $user_name . '";
    $query = $database->prepare($sql);
    try{
        $query->execute();
    } catch (PDOException $fehler) {
        Session::add('response_negative', 'Es ist ein Fehler aufgetreten. ');
    }
    return $query->fetchAll();
}
```

Abbildung 102: getSpecificStudent2-Funktion aus NotenModel

Die View stellt die Informationen über die Noten in den einzelnen Fächern als Tabelle dar. Hierzu wird die Table-Klasse verwendet. Mit dem „Zeugnis/Notenspiegel drucken“-Button kann daraufhin die Notenliste gedruckt werden. Hierzu wird dann das TCPDF-Tool verwendet.

```
<article>
    <h1>Zeugnis/Notenspiegel</h1>
    <!-- Sobald der Button geklickt wird, wird die printZeugnis-funktion ausgeführt-->
    <form action="index.php?url=zeugnis/printZeugnis" method="post" target="_blank" />
        <?php
            //das übergebene Array noten_list wird durch die Table-Klasse in eine Tabelle umgewandelt und angezeigt.
            if ($this->noten_list) {
                $table = new Table ();
                $tableInfo = array (
                    "0" => "Veranstaltungsnummer",
                    "1" => "Veranstaltungsbezeichnung",
                    "2" => "Note"
                );
                $table->table ( array (
                    'table' => $this->noten_list,
                    'alias' => array (
                        $tableInfo
                    )
                ) );
            } else {
                Session::add ( 'response_negative', 'Es ist ein Fehler aufgetreten. Die Notenliste konnte nicht erstellt werden.' );
            }
            echo '<input class="button" type="button" value="Zurück" onClick="history.back();">';
        ?>
        <input class="button" type="submit" id="b1" value="Zeugnis/Notenspiegel drucken" ></a>
    </form>
</article>
```

Abbildung 103: zeugnisView

Die printZeugnis-Funktion des Controllers, welche ausgeführt wird sobald der Button geklickt wird, stellt sich wie folgt dar.

```
public function printZeugnis() {
    /*
        nutzt das TCPDF Tool im Ordner ../application/lib um eine PDF zu erstellen.
        Inhalt ist die Variable $html.
    */
    $current_user = Session::get('user_name');
    $studiengang = UserModel::getStudentStudiengang($current_user);
    $timestamp = time();
    $date = date("d.m.Y", $timestamp);

    //Studentendaten aus der Datenbank auslesen und in Variablen speichern um sie im Notenspiegel anzuzeigen.
    $studentinfo = UserModel::getUserData5($current_user);
    foreach ( $studentinfo as $key => $value ) {
        $nachname = htmlentities ( $value->nachname );
        $vorname = htmlentities ( $value->vorname );
        $straßenname = htmlentities ( $value->straßenname );
        $hausnummer = htmlentities ( $value->hausnummer );
        $plz = htmlentities ( $value->plz );
        $stadt = htmlentities ( $value->stadt );
        $land = htmlentities ( $value->land );
    }
}
```

Abbildung 104: printZeugnis-Funktion aus ZeugnisController Teil 1

Zunächst wird dafür gesorgt, dass alle nötigen Informationen vorhanden sind. Für einen Notenspiegel sind die Kontaktdata des Studenten, ein Datum, sowie der Studiengang nötig. Diese Informationen erhält der Controller über den Gebrauch der getStudentStudiengang-Funktion und der getUserData5-Funktion des UserModels.

Daraufhin werden verschiedene Formatierungen und Einstellungen für das TCPDF-Tool getroffen, welche an dieser Stelle jedoch nicht abgebildet sind. (vgl. „Nutzerliste drucken“)

```

$html = 'Lehrveranstaltungsmanagementsystem, '.$date.'<br><h1>Notenspiegel/Zeugnis</h1><br><p style="font-size: 14px;">';
$html .= '$vorname.' . $nachname.'<br>' . $strasseName.' '. $hausnummer.'<br>' . $plz.' '. $stadt.'<br>' . $land.'</p>';
foreach ( $studiengang as $key => $value ) {
    $html = $html.'<h3>Studiengang: ' . htmlentities( $value->stg_bezeichnung ) . '</h3>';
    $html = $html.'<table border="1" cellpadding="1"><tr align="center">
        <th><b>Veranstaltungsnr.</b></th><th><b>Veranstaltungsbezeichnung</b></th><th><b>Note</b></th></tr>';
    $data = array ('noten_list' => NotenModel::getSpecificStudents2($current_user));
    foreach ( $data as $key => $value ) {
        $this->{$key} = $value;
    }
    foreach($this->noten_list as $key => $value){
        $html = $html.'<tr><td>' . htmlentities($value->veranst_ID) . '</td>';
        $html = $html.'<td>' . htmlentities($value->veranst_bezeichnung) . '</td>';
        $html = $html.'<td>' . htmlentities($value->note) . '</td>';
        $html = $html.'</tr>';
    }
    $html = $html.'</table>';
}
$pdf->writeHTML ($html, $ln=true, $fill=false, $reseth=false, $cell=false, $align='');
$pdf->lastPage();
//Output der PDF
$pdf->Output('notenspiegel.pdf', 'I');

```

Abbildung 105: printZeugnis-Funktion aus ZeugnisController Teil 2

In Bild 2 zur printZeugnis-Funktion sieht man wie die zuvor gesammelten Informationen in einen HTML-Code eingebaut werden. Zudem wird die Notenliste als Tabelle hinzugefügt.

Der HTML-Code wird darauf hin durch die writeHTML-Methode in die PDF-Datei geschrieben. Durch den Aufruf der Output-Methode wird die PDF-Datei dann ausgegeben. In diesem Fall wird diese dann in einem neuen Tab im Browser ausgegeben. Von dort aus kann sie weiter verarbeitet, gespeichert oder gedruckt werden.

5.21 . Nutzerliste drucken

Für einen Mitarbeiter oder Administrator, der mit dem Lehrveranstaltungsmanagementsystem arbeitet, ist es eine sinnvolle Funktion eine Nutzerliste drucken zu können, welche sämtliche Nutzer enthält. Hierzu gibt es die User Story „Als Admin möchte ich alle Nutzer in einer Liste speichern können. (Nr. 280)“.

Durch folgende Funktion im UserModel werden die Nutzerdaten mit allen wichtigen Informationen abgefragt.

```

public function getUserDataAll4 () {
    $database = DatabaseFactory::getFactory()->getConnection ();
    $sql = "Select u.nachname, u.vorname, u.nutzer_name, rolle_bezeichnung, email_name, straßenname, hausnummer, plz, stadt, land
            from Nutzer u
            join Rolle r on r.rolle_ID = u.rolle_ID
            join EMail e on e.nutzer_name = u.nutzer_name
            join Adresse a on a.nutzer_name = u.nutzer_name
            order by nachname asc";
    $query = $database->prepare ($sql);
    try{
        $query->execute ();
        return $query->fetchAll ();
    } catch(PDOException $e){
        Session::add ( 'response_warning', 'Es ist ein Fehler bei der Datenbankabfrage aufgetreten.' );
    }
}

```

Abbildung 106: getUserDataAll4 aus UserModel

Diese Informationen werden der View vom Controller übergeben und mit der Table-Klasse zu einer ansehnlichen Tabelle verarbeitet.

```

if ($this->user_list) {
    $table = new Table ();
    $userInfo = array (
        "0" => "Nachname",
        "1" => "Vorname",
        "2" => "Nutzername",
        "3" => "Rolle",
        "4" => "E-Mail",
        "5" => "Straßenname",
        "6" => "Hausnummer",
        "7" => "PLZ",
        "8" => "Stadt",
        "9" => "Land"
    );
    $table->table ( array (
        'table' => $this->user_list,
        'alias' => array (
            ... $userInfo
        )
    ) );
}

```

Abbildung 107: nutzerlisteDruckenView

Nutzerliste

Im Folgenden sehen Sie eine Liste von allen Nutzern:

Nachname	Vorname	Nutzername	Rolle	E-Mail	Straßenname	Hausnummer	PLZ	Stadt	Land
Abdu	Kasimir	kaabdu	Student	kaabdu@hs-ulm.de	Hauptstraße	4		89233 Neu-Ulm	Deutschland
Ali	Ahmed	ahali	Student	ahali@hs-ulm.de	Kirschenweg	12		89233 Neu-Ulm	Deutschland
Beck	Jonas	jobeck	Student	jobeck@hs-ulm.de	Mirabellenweg	6		89233 Neu-Ulm	Deutschland
Brook	Kelly	kebroo	Student	kebroo@hs-ulm.de	Steingrube	6		89231 Neu-Ulm	Deutschland
Carr	Nicholas	nicarr	Student	nicarr@hs-ulm.de	Leipheimer Straße	1		89233 Neu-Ulm	Deutschland
Danzer	Lars	ladanz	Student	ladanz@hs-ulm.de	Eulesweg	4		89231 Neu-Ulm	Deutschland
Doll	Carson	cadoll	Student	cadoll@hs-ulm.de	Turmstraße	16		15028 Berlin	Deutschland
Doof	Moritz	modoof	Student	modoof@hs-ulm.de	Frauenstraße	113		89073 Ulm	Deutschland
Duck	Derek	deduck	Student	deduck@hs-ulm.de	Olivenweg	45		89073 Ulm	Deutschland
Etoo	Samuel	saetoo	Student	saetoo@hs-ulm.de	Bahnofgasse	15		89075 Ulm	Deutschland
Firmino	Roberto	mitarbeiter	Mitarbeiter	mitarbeiter@hs-	Traminoweg	25		89075 Ulm	Deutschland

Abbildung 108: Nutzerliste Darstellung

Nun soll diese Liste gedruckt bzw. gespeichert werden können. Hierzu wird das TCPDF-Tool in der printUser-Funktion des Controllers genutzt.

```
public function printUser() {
    /*
        nutzt das TCPDF Tool im Ordner ../application/lib um eine PDF zu erstellen.
        Inhalt ist die Variable $html.
    */
    $timestamp = time();
    $date = date("d.m.Y", $timestamp);
    require_once('../application/lib/tcpdf/tcpdf.php');
    ob_start();
    $pdf = new TCPDF("L", "mm", "A4", true, "UTF-8", true);
    $pdf->SetCreator("PDF_Creator");
    $pdf->SetAuthor("Admin/Mitarbeiter");
    $pdf->SetMargins(25, 25, 25, true);
    $pdf->SetAutoPageBreak(true, 25);
    $pdf->SetPrintHeader(false);
    $pdf->SetPrintFooter(false);
    $pdf->SetFont('helvetica', '', 10);
    $pdf->AddPage();
```

Abbildung 109: printUser-Funktion des nutzerlisteDruckenControllers Teil 1

In Teil 1 der Funktion werden verschiedene Einstellungen und Formatierungen vorgenommen. Teil 2 befasst sich mit dem Inhalt der PDF. Hierzu wird die Variable \$html genutzt. Diese umfasst sämtlichen Inhalt als HTML-Code, welcher dann mit der writeHTML-Methode in die PDF geschrieben wird.

```

$html = 'Lehrveranstaltungsmanagementsystem, '.date().'<h1>Nutzerliste</h1><br><table border="1" cellpadding="1"><tr align="center">
|<th><b>Nachname</b></th><th><b>Vorname</b></th><th><b>Nutzernname</b></th><th><b>Rolle</b></th>
|<th><b>E-Mail</b></th><th><b>Straßenname</b></th><th><b>Haus-Nr.</b></th><th><b>PLZ</b></th><th><b>Stadt</b></th><th><b>Land</b></th>
|</tr>';
$data = array ('user_list' => UserModel::getUserDataAll4 ());
foreach ( $data as $key => $value ) {
    $this->{$key} = $value;
}
foreach ($this->user_list as $key => $value) {
    $html = $html.'<tr><td>'.htmlentities($value->nachname).'</td>';
    $html = $html.'<td>'.htmlentities($value->vorname).'</td>';
    $html = $html.'<td>'.htmlentities($value->nutzer_name).'</td>';
    $html = $html.'<td>'.htmlentities($value->rolle_bezeichnung).'</td>';
    $html = $html.'<td>'.htmlentities($value->email_name).'</td>';
    $html = $html.'<td>'.htmlentities($value->straßenname).'</td>';
    $html = $html.'<td>'.htmlentities($value->hausnummer).'</td>';
    $html = $html.'<td>'.htmlentities($value->plz).'</td>';
    $html = $html.'<td>'.htmlentities($value->stadt).'</td>';
    $html = $html.'<td>'.htmlentities($value->land).'</td>';
    $html = $html.'</tr>';
}
$html = $html.'</table>';

$pdf->writeHTML ($html, $ln=true, $fill=false, $reseth=false, $cell=false, $align='');

$pdf->lastPage();
//Output der PDF
$pdf->Output('nutzerliste.pdf', 'I');

```

Abbildung 110: printUser-Funktion des nutzerlisteDruckenControllers Teil 2

Die PDF, welche mit dem Tool dadurch ausgegeben wird, sieht dann folgendermaßen aus. In diesem Fall wird der Output bzw. die PDF in einem neuen Tab des Browsers angezeigt. Von dort aus kann die Liste gedruckt oder abgespeichert werden.

Lehrveranstaltungsmanagementsystem, 02.07.2015									
Nutzerliste									
Nachname	Vorname	Nutzernname	Rolle	E-Mail	Straßenname	Haus-Nr.	PLZ	Stadt	Land
Abdu	Kasimir	kaabdu	Student	kaabdu@hs-ulm.de	Hauptstraße	4	89233	Neu-Ulm	Deutschland
Ali	Ahmed	ahali	Student	ahali@hs-ulm.de	Kirschenweg	12	89233	Neu-Ulm	Deutschland
Beck	Jonas	jobeck	Student	jobeck@hs-ulm.de	Mirabellenweg	6	89233	Neu-Ulm	Deutschland
Brook	Kelly	kebroo	Student	kebroo@hs-ulm.de	Steingrube	6	89231	Neu-Ulm	Deutschland
Carr	Nicholas	nicarr	Student	nicarr@hs-ulm.de	Leipheimer Straße	1	89233	Neu-Ulm	Deutschland
Danzer	Lars	ladanz	Student	ladanz@hs-ulm.de	Eulerweg	4	89231	Neu-Ulm	Deutschland
Doll	Carson	cadoll	Student	cadoll@hs-ulm.de	Turmstraße	16	15028	Berlin	Deutschland
Doof	Moritz	modoof	Student	modoof@hs-ulm.de	Frauenstraße	113	89073	Ulm	Deutschland
Duck	Derek	deduck	Student	deduck@hs-ulm.de	Olivenweg	45	89073	Ulm	Deutschland

Abbildung 111: Nutzerliste - Output des PDF Tools

5.22 E-Mails versenden

In einem Lehrveranstaltungssystem sollte es grundsätzlich eine einfache Möglichkeit geben, verschiedene Personen per Email zu kontaktieren. Da dies eine der wichtigsten Formen des Kommunikationsaustausches an einer Hochschule darstellt, wurde hierfür ebenfalls eine eigene User Story erstellt:

„Nr.: xxx, Als Dozent möchte ich Emails an alle Teilnehmer einer Veranstaltung schicken können“

Für diese User Story wurde eine externe Quelle verwendet. Hierbei handelt es sich um „**PHPMailer**“, dass auf folgender Internetseite heruntergeladen werden kann:

<https://github.com/PHPMailer/PHPMailer>

Die externen Klassen wurden in das Projekt integriert und konnten somit verwendet. Hier wurden zwei Möglichkeiten bereitgestellt eine E-Mail zu versenden. Zum einen in einem eigenständigen Menü für alle Rollen und zum anderen als Sammelmail an alle Teilnehmer einer Vorlesung für Dozenten. Das folgende Schaubild zeigt die erste Möglichkeit.

The screenshot shows a web-based application interface for sending emails. At the top, there is a horizontal navigation bar with links: Home, Mein Profil, Raumplan, Stundenplan, and Teilnehmerliste. Below this, the main content area has a title 'Email versenden'. The form fields are as follows:

- Ihr Name:
- Empfänger: * (with value 'test@beispiel.com')
- Betreff: *
- Nachricht: * (with value 'Hallo')
- Anhang: Keine Dateien ausgewählt.
Dateiformat: png, jpg, jpeg, gif, txt, htm, pdf, zip - Dateigröße max.: 1,0 MB
-

At the bottom of the form, there is a note: 'Bitte alle mit * markierten Felder ausfüllen.'

Abbildung 112: Ansicht View E-Mail versenden

In der Folgenden Abbildung wird die zweite Möglichkeit dargestellt.

The screenshot shows a web-based application for managing course schedules and participant lists. At the top, there is a navigation bar with links: Home, Mein Profil, Raumplan, Stundenplan, and Teilnehmerliste. Below the navigation bar, the main content area has a title: "Email an alle Teilnehmer des Kurses "Technologische Grundlagen" versenden". The form fields include: Betreff (Subject) with placeholder "Betreff" and an asterisk indicating it is required; Nachricht (Message) with placeholder "Hallo" and an asterisk; Anhang (Attachment) with a browse button "Durchsuchen...", a message "Keine Dateien ausgewählt", and a link "Anhang löschen"; and a note about file formats: "Dateiformat: png, jpg, jpeg, gif, txt, htm, pdf, zip - Dateigröße max.: 1,0 MB". A large blue button labeled "Email senden" (Send email) is prominently displayed. Below the form, a note says: "Bitte alle mit * markierten Felder ausfüllen." (Please fill in all fields marked with *). At the bottom left is a link "Zurück Auswahl" (Back to selection), and at the bottom right is a link "copyright und andere informationen über diese seite" (Copyright and other information about this page).

Abbildung 113: View Email an alle Teilnehmer versenden

Die zweite Möglichkeit verschickt eine E-Mail an alle Teilnehmer einer Veranstaltung. In dieser Darstellung besteht die Möglichkeit wieder zurück zur Teilnehmerliste zu kehren, indem auf den Button „Zurück zur Auswahl“ geklickt wurde. Der Hauptteil des Programmcodes befindet sich in der „EmailController.php“ Datei. In der Folgenden Abbildung ist der Quellcode zum Versenden einer Email dargestellt. Dort werden die einzelnen Parameter an die Klassenattribute übergeben und mit der Funktion Send versendet.

```

public function sendMailDozent(){
    $mail = new PHPMailer(true);
    $mail->IsSMTP(); //Versand über SMTP festlegen
    try{
        $mail->Host = Config::get('smtp_host'); //SMTP-Server setzen
        $mail->SMTPDebug = 0; // Debug information
        $mail->SMTPAuth = true; //Authentifizierung aktivieren
        $mail->Port = Config::get('smtp_port');
        $mail->Username = Config::get('smtp_user'); // SMTP Benutzername
        $mail->Password = Config::get('smtp_pass'); // SMTP Passwort
        $mail->AddAddress(Request::post('email'));
        $mail->AddReplyTo = Config::get('smtp_email');
        $mail->From = Config::get('smtp_email');
        $mail->FromName = Request::post('name');
        $mail->AddBCC('wysdam@googlemail.com');
        $validAttachments = array();
        foreach($_FILES['File']['name'] as $index => $fileName)
        {
            $filePath = $_FILES['File']['tmp_name'][$index];
            if(is_uploaded_file($filePath))
            {
                $attachment = new stdClass;
                $attachment->fileName = $fileName;
                $attachment->filePath = $filePath;
                $validAttachments[] = $attachment;
            }
        }
        foreach($validAttachments as $attachment)
        {
            $mail->AddAttachment($attachment->filePath, $attachment->fileName);
        }
        $mail->Subject = Request::post('betreff');
        $mail->Body = Request::post('nachricht')."\n\n Viele Grüsse\n\n".Request::post('name');

        if(!$mail->Send())
        {
            Session::add('response_negative', 'Email wurde nicht versandt. Es ist ein Fehler aufgetreten: '.$mail->ErrorInfo);
        }
        else
        {
            Session::add('response_positive', 'Email wurde versandt.');
        }
        $this->View->render ('dozent/teilnehmerListe', array (
            'teilnehmer' => DozentModel::getTeilnehmer(Request::post ('id'))
        ));
    }catch (phpmailerException $e){
        echo $e->getMessage();
    }
}

```

Abbildung 114: Controller Versendet eine Email

Sobald eine E-Mail erfolgreich versendet wurde, wird eine positive Rückmeldung gegeben. Ansonsten wird eine Fehlermeldung zurückgegeben.

5.23 Informationen zum Studiengang anzeigen

Die User Story „Als Student möchte ich online Infos zu meinem Studiengang einsehen können. (Nr. 470)“ befasste sich mit der Anforderung, dass das Lehrveranstaltungsmanagementsystem auch Informationen über den Studiengang des aktuell eingeloggten Studenten anzeigen soll. Studieninhalte, welche meistens als Bilddatei zur Verfügung stehen, und eventuell ein kurzer Informationstext sollen dabei angezeigt werden.

Zunächst muss der Controller also abfragen, wer zur Zeit eingeloggt ist. Dies geschieht über die Session::get-Methode, welche den Username des aktuell eingeloggten Studenten zurück gibt. Mit dieser Information wird die View erzeugt.

```
class informationController extends Controller {
    public function __construct() {
        parent::__construct ();
    }

    //ruft die informationView auf und übergibt dazu die Information welche/n Studiengang/Studiengänge der Student studiert
    public function information() {
        $current_user = Session::get('user_name');
        $this->View->render ('information/information', array (
            'studiengang' => UserModel::getStudentStudiengang($current_user)
        ) );
    }
}
```

Abbildung 115: informationController

Dabei wird über die getStudentStudiengang-Funktion im UserModel die Information überreicht, welchen Studiengang der aktuelle User studiert.

```
public function getStudentStudiengang($user_name) {
    $database = DatabaseFactory::getFactory()->getConnection ();
    $sql = "Select stg_bezeichnung
            from studiengang s
            join student n on s.stg_ID = n.stg_ID where n.nutzer_name ='".$user_name."'";
    $query = $database->prepare ($sql);
    try{
        $query->execute ();
        return $query->fetchAll ();
    } catch(PDOException $e){
        Session::add ('response_negative', 'Es ist ein Fehler bei der Datenbankabfrage aufgetreten.' );
    }
}
```

Abbildung 116: getStudentStudiengang aus UserModel

Die View erzeugt aus dieser Information eine Seite, die aus einem Bild, eventuell einem Text, sowie einem „zurück“-Button besteht. Das Bild, welches nach dem Studiengang benannt im untergeordneten Content-Ordner liegen muss, sollte einen Überblick über die Studieninhalte schaffen.

```

<article>
    <h1>Studiengang-Information</h1>
    <?php
        // Übergebenes Array studiengang (studiengang des aktuell eingeloggten Studenten) wird genutzt um Studienganginformation anzuzeigen
        if ($this->studiengang) {
            foreach ( $this->studiengang as $key => $value ) {
                // Studiengang als Überschrift
                echo '<h3>';
                echo htmlentities ( $value->stg_bezeichnung );
                echo '</h3>';
                // Studienganginformationen in Form von Bild und Text anzeigen
                $stg_bezeichnung = htmlentities ( $value->stg_bezeichnung );
                echo '<p><img alt="'.APPLICATION_PATH.'/view/information/Content/'.$stg_bezeichnung.'.jpg" alt="Studienganginformation '.$stg_bezeichnung.'"></p>';
                /* Optional kann auch ein Text (.txt-Datei) eingeschlossen werden
                * echo '<p><object data="'.APPLICATION_PATH.'/view/information/Content/'.$stg_bezeichnung.'.txt" type="text/plain" width=100% height=500px">
                </object></p><br>';
                */
            }
        }
        echo '<br><input class="button" type="button" value="Zurück" onClick="history.back();">';
    ?>
</article>

```

Abbildung 117: informationView

Dies kann für einen Studenten der Medizintechnik beispielsweise folgende Ansicht annehmen.

The screenshot shows a web page titled "Studiengang-Information" with a sub-section titled "Medizintechnik". Below this, there is a grid-based course schedule for the "Medizintechnik" program. The grid has columns for "Sem.", "Wahl-Fach 3", "Wahl-Fach 4", "BWL", and "Abschlussarbeit". Rows represent semesters from 1 to 7. Semester 1 includes courses like Mathe1, Physik1, MED1, Elektron. Schaltgn, Mikrocontroller, and Software Entwicklg. Semester 2 includes Mathe2, Physik2, MED2, Analog-technik1, Embedded Systems1, and Betriebs-systeme. Semester 3 includes Mathe3, Signale + Systeme, Bildgebende Verfahren, Elektron. Schaltgn, Mikrocontroller, and Software Entwicklg. Semester 4 includes Wahl-fach1, Digitale Signal-verarbeitg, klinische Medizin, Analog-technik1, Embedded Systems1, and Betriebs-systeme. Semester 5 is labeled "Praxissemester". Semester 6 includes Wahl-Fach2, Medizin. Elektronik, Bild-gestützte Medizin, Sensorik1, Kommun.-technik, and Software-Engineerg. Semester 7 includes Wahl-Fach3, BWL, and Abschlussarbeit. At the bottom of the grid, there are labels for subjects: Physik, Mathematik, Signale, Medizin, -technik, Elektronik, Digital-technik, Prakt. Informatik, and Ingenieurs-informatik. A "Zurück" button is located at the bottom left of the grid area.

Sem.	Wahl-Fach 3	Wahl-Fach 4	BWL	Abschlussarbeit		
7						
6	Wahl-Fach 2	Medizin. Elektronik	Bild-gestützte Medizin	Sensorik 1	Kommun.-technik	Software-Engineerg
5	Praxissemester					
4	Wahl-fach 1	Digitale Signal-verarbeitg	klinische Medizin	Analog-technik 1	Emddedded Systems 1	Betriebs-systeme
3	Mathe 3	Signale + Systeme	Bildgebende Verfahren	Elektron. Schaltgn	Mikro-controller	Software Entwicklg.
2	Mathe 2	Physik 2	MED 2	Elektrotechnik 2		Obj-orient. Program'ng
1	Mathe1	Physik 1	MED 1	Elektro-technik 1	Digital-technik	Prakt. Informatik
	Physik, Mathematik, Signale	Medizin, -technik	Elektronik	Digitale Systeme	Ingenieurs-informatik	

Abbildung 118: Ansicht Informationen Medizintechnik

6. Abbildungsverzeichnis

- Abbildung 1: Ablauf Scrum
(<https://www.mountaingoatsoftware.com/system/asset/file/17/ScrumLargeLabelled.png>)
- Abbildung 2: Burndown Chart Sprint 1
- Abbildung 3: Burndown Chart Sprint 2
- Abbildung 4: Burndown Chart Sprint 3
- Abbildung 5: Burndown Chart Sprint 4
- Abbildung 6: Burndown Chart Sprint 5
- Abbildung 7: Burndown Chart Sprint 6
- Abbildung 8: Projekt Burndown Chart
- Abbildung 9: ER-Modell Teil 1
- Abbildung 10: ER-Modell Teil 2
- Abbildung 11: Verzeichnis
- Abbildung 12: Dateiverzeichnis auf GitHub
- Abbildung 13: MVC
- Abbildung 14: Aufbau der View
- Abbildung 15: Ansicht - Erfolgreich Eingeloggt
- Abbildung 16: Table-Aufruf ohne Alias und Link
- Abbildung 17: Table-Aufruf mit Alias und mit Link
- Abbildung 18: Table-Aufruf mit Alias und mit Link
- Abbildung 19: Ansicht Login-Fenster
- Abbildung 20: Ansicht im eingeloggten Zustand
- Abbildung 21: Error Controller - Seite nicht gefunden
- Abbildung 22: CSS – Header
- Abbildung 23: CSS-Button
- Abbildung 24: CSS-Button (hover)
- Abbildung 25: CSS – Navigation
- Abbildung 26: View Studenten anlegen
- Abbildung 27: Controller Student anlegen

Abbildung 28: Letzte aktuelle Matrikelnummer extrahieren

Abbildung 29: Alle hinterlegten Studiengänge auslesen

Abbildung 30: View Anlegen eines Studenten

Abbildung 31: View Studiengänge befüllen

Abbildung 32: Controller Student anlegen

Abbildung 33: Model Nutzernamen bestimmen

Abbildung 34: Model Nutzerdaten speichern

Abbildung 35: Model Adressdaten speichern

Abbildung 36: Model Emaildaten speichern

Abbildung 37: Model Studentendaten speichern

Abbildung 38: Erfolgreiche Rückgabe bei Anlage eines Studenten

Abbildung 39: Ansicht Veranstaltung anlegen

Abbildung 40: anlegenForm-Funktion im VeranstaltungController

Abbildung 41: getVeranstaltungsart

Abbildung 42: getAusstattung

Abbildung 43: getStudiengaenge

Abbildung 44: Veranstaltung anlegen View Teil 1

Abbildung 45: Veranstaltung anlegen View Teil 2

Abbildung 46: raumAnlegenView

Abbildung 47: Startseite Raum anlegen

Abbildung 48: raumSelected-Funktion raumAnlegenController

Abbildung 49: raumSelected-Funktion raumAnlegenController Fehlerbehandlung

Abbildung 50: Darstellung vorlesungsraumAnlegenView Abbildung 51: Darstellung laborraumAnlegenView

Abbildung 53: Darstellung bibliothekAnlegenView

Abbildung 54: vorlesungsraumAnlegenView

Abbildung 55: raumStammdatenView

Abbildung 56: getGebäude raumAnlegenModel

Abbildung 57: getAusstattung raumAnlegenModel

Abbildung 58: raumAusstattungView

Abbildung 59: getBuchkategorien raumAnlegenModel

Abbildung 60: bibliothekAnlegenView Darstellung Buchkategorien

Abbildung 61: saveRaum-Funktion raumAnlegenController

Abbildung 62: raumStammdaten-Funktion im raumAnlegenModel

Abbildung 64: ausstattungVorlesungsraum-Funktion im raumAnlegenModel

Abbildung 63: vorlesungsraumAnlegen-Funktion im raumAnlegenModel

Abbildung 65: Ansicht - Vorlesungsraum erfolgreich angelegt

Abbildung 66: studenplanController, Funktion studenplan_individuell()

Abbildung 67: studenplanModel, Funktion getVeranstaltungstermine_individuell()

Abbildung 68: Studentenstundenplan

Abbildung 69: Formular zur Auswahl eines Raumes

Abbildung 70: raumplanController, Funktion erzeugeFormular()

Abbildung 71:raumplanController, Funktion raumplanAnzeigen()

Abbildung 72: Raumplan

Abbildung 73: View Auswahl einer Vorlesung

Abbildung 74: View Teilnehmerliste für die Vorlesung TEGR

Abbildung 75: Profil eines Dozenten anzeigen und bearbeiten

Abbildung 76: Ordnerstruktur für das Projekt

Abbildung 77: Model zur Abfrage der Profildaten

Abbildung 78: Controller zur Steuerung der View

Abbildung 79: View Quellcode der Anzeige

Abbildung 80: Controller Aufruf bei Ändern eines Attributs

Abbildung 81: Model Straße eines Dozenten ändern

Abbildung 82: Positive Fehlermeldung nach Änderung

Abbildung 83: View Auswahl Kurs oder Student

Abbildung 84: View Note für Teilnehmer ändern

Abbildung 85: Veranstaltung bearbeiten – View

Abbildung 86: Veranstaltung als Wahlfach – View – wahlfachVeranstaltungSelect

Abbildung 87: Veranstaltung als Wahlfach – Controller – wahlfachVeranstaltungSelect

Abbildung 88: Darstellung Auswahl der Erweiterung

Abbildung 89: typSelected aus veranstaltungErweiternController

Abbildung 90: umPersonErweiternView Teil 1

Abbildung 91: umPersonErweiternView Teil 2

Abbildung 92: umPersonErweitern Teil 3

Abbildung 93: umPersonErweiternView

Abbildung 95: selected-Funktion im veranstaltungErweiternController Teil 2

Abbildung 94: selected-Funktion im veranstaltungErweiternController Teil 1

Abbildung 96: pruefenDozent aus veranstaltungErweiternModel

Abbildung 97: setErweiterungStudent aus veranstaltungErweiternModel

Abbildung 98: setErweiterungDozent aus veranstaltungErweiternModel

Abbildung 99: Erfolgreiche Erweiterung Dozent

Abbildung 100: Erfolgreiche Erweiterung Student

Abbildung 101: Konstuktor zeugnisController

Abbildung 102: getSpecificStudent2-Funktion aus NotenModel

Abbildung 103: zeugnisView

Abbildung 104: printZeugnis-Funktion aus ZeugnisController Teil 1

Abbildung 105: printZeugnis-Funktion aus ZeugnisController Teil 2

Abbildung 106: getUserDataAll4 aus UserModel

Abbildung 107: nutzerlisteDruckenView

Abbildung 108: Nutzerliste Darstellung

Abbildung 109: printUser-Funktion des nutzerlisteDruckenControllers Teil 1

Abbildung 110: printUser-Funktion des nutzerlisteDruckenControllers Teil 2

Abbildung 111: Nutzerliste - Output des PDF Tools

Abbildung 112: Ansicht View E-Mail versenden

Abbildung 113: View Email an alle Teilnehmer versenden

Abbildung 114: Controller Versendet eine Email

Abbildung 115: informationController

Abbildung 116: getStudentStudiengang aus UserModel

Abbildung 117: informationView

Abbildung 118: Ansicht Informationen Medizintechnik

Abbildung 52: Darstellung büroAnlegenView

7. Eigenarbeitserklärung

Hiermit erklären wir, dass wir die vorliegende Dokumentation selbstständig verfasst haben.

Zu jeder Abbildung, ob aus dem Projekt oder aus externen Quellen, sind in der Bezeichnung Angaben zur Herkunft gemacht.

Unterschrift Kris Klamser

Unterschrift Kilian Kraus

Unterschrift Alexander
Mayer

Unterschrift Roland Schmid

Unterschrift Damian Wysocki

8. Anlagen

Als Anlagen angefügt ist eine CD-Rom, welche die komplette Software nach aktuellem Stand (23.7.2015) enthält. Ebenfalls auf der CD zu finden sind die Präsentation zu diesem Projekt sowie die Dokumentation in PDF-Format.