Projektarbeit

Fakultät Elektronik und Informatik

Ein webbasiertes Kurswahlsystem für die Schillerschule Aalen

Betreuer: Prof. Dr. Ulrich Klauck

Michael Schütz und Christian Silfang

Wintersemester 2012/2013

Zusammenfassung

Diese Projektarbeit beschäftigt sich mit dem Thema der Modellierung und Implementierung eines Kurswahlsystems für eine Werksrealschule.

Die Modellierung beschäftigte sich vor allem mit dem Erarbeiten von Lösungsansätzen zur Datenhaltung und Umsetzungen in den Webtechnologien unter Beachtung der aktuell geltenden Standards der Softwareentwicklung.

Das Resultat der Implementierung des Projekts ist eine Webanwendung welche mit Hilfe des Apache Webstandards, den Java Server Faces umgesetzt wurde. Ferner wurde eine komplette Laufzeitumgebung für die bereits bestehende Infrastruktur der Schule erarbeitet und eingerichtet.

Neben der eigentlichen Ausarbeitung werden ebenfalls nötige Grundlagen besprochen. Somit sind für die Ausarbeitung keine Vorkenntnisse nötig, jedoch durchaus hilfreich.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Im Sinne unserer Medien-Projektarbeit haben wir ein webbasiertes System zur Kurswahl und Kursverwaltung der Aalener Werkrealschule 'Schillerschule' [1] realisiert.

Ein kurzes Portrait der Schillerschule ([aal13]):

> *"Die Schillerschule ist eine städtische Ganztagesschule mit rund 500 Schülerinnen und Schülern. Die Entwicklung zu einem Lern- und Lebensraum kennzeichnet das Profil der Schule. Wesentliche Ziele sind die Förderung von Lernen und Leistung sowie die kulturelle und gesellschaftliche Integration der Kinder und Jugendlichen mit ihren unterschiedlichen Herkunfts- und Erfahrungshorizonten. „*

Wichtig war es, dass das Kurswahlsystem von verschiedenen Benutzern, wie Schülern sowie Lehrern, von überall aus zugänglich ist und außerdem später von einem Administrator verwaltet werden kann. Ferner sollte das gesamte Kurswahlmanagment über das neue Tool 'KuWaSys' erfolgen.

Die Realisierung des Systems wurde serverseitig mit einem Apache Tomcat und Java Server Faces (JSF), genauer mit einer Implementierung der JSF namens JSF-myFaces 2.1 und JSF-myFaces Tomahawk in der Version 2.0, umgesetzt. Die Zugänglichkeit auf der Seite der Clients konnte somit über jeden beliebigen Webbrowser realisiert werden.

Die benötigte Haltung der Daten sowie deren Bereitstellung wurde mit einer PostgreSQL Datenbank (DB) für das gesamte System umgesetzt.

[1] http://www.schillerschule-aalen.de

Das User Interface (UI) wurde mit HTML, genauer gesagt mit HTML 5 umgesetzt. Der HTML Code wurde nicht wie es in der serverseitigen Programmierung üblich ist in Java Server Faces (JSF)-Dateien eingebettet, sondern in eigenständige HTML Dateien. Diese werden in der JSF-Technologie kurz: 'Facelets' genannt.

1.2 Problemstellung und -abgrenzung

Die Schillerschule hat bereits ein 'in die Jahre gekommenes' Tool zur Realisierung der Kurswahlen von Schülern, das bis dahin genannte 'KuhWa-Tool'. Ebenso wird momentan die Verwaltung der Kurse, von Lehrern der Schule über dieses Tool abgewickelt. Die Schwierigkeiten welche durch das alte Tool entstanden sind können aufgrund ihrer Beschaffenheit in zwei grundlegende Kategorien eingeteilt werden:

Das umständliche und nicht intuitive gestaltete User Interface (UI) auf der einen Seite, die mitlerweile überholte Programmstruktur mit diverser mangelhafter Funktionalität auf der Anderen. Die Bedienung des Tools erweißt sich als nicht mehr praktikabel, da es erstens kompliziert zu verwalten ist und nur geschultes Lehrpersonal es bedienen kann, zweitens ist das Kurswahlsystem nur auf einer Laufzeitumgebung der Schule problemlos lauffähig und einsetzbar. Probleme die auf die Funktionalität der Software zurückzuführen sind und darüber hinaus für die Schule einen weitaus größeren Mehraufwand bedeuteten, haben jedoch ein weitaus größere Bedeutung für die Kurswahl der Schule, wie nur das umständlich zu bedienende Design.

Das alte Tool hat keine Möglichkeiten für die Benutzerdatenverwaltung der Schüler und Lehrer der Schule. Änderungen sind schwer wieder rückgängig zu machen, Systemänderungen teilweise garnicht. Ein Mehrbenutzerbetrieb ist weder für Kurswahlen noch für die Verwaltung möglich. Ein einheitliches Konzept für die DB ist ebenso nicht vorhanden. Bei der bisher verwendeten DB handelt es sich bisher um eine lokal angelegte Microsoft Access Datenbank. Der Zugriff auf diese ist also auch nur systemweit

möglich.

1.3 Über diese Ausarbeitung

Sinn und Zweck der Ausarbeitung

Es soll zu Beginn erwähnt werden, dass diese Ausarbeitung nicht als eine Anleitung oder eine vollständige Dokumentation unserer Implementierung verstanden werden soll. Wobei der zweite Punkt treffender wäre. Wir wollen vor allem Einblicke in die Arbeit mit JSF, der dazugehörigen Umgebung und die Umsetzung eines Projekts der Informationstechnik (IT) dieser Größenordnung geben. Ferner wollen wir einen strukturierten und nachvollziehbaren Verlauf unserer Entwicklungen darstellen.

Die Dokumentation des in JAVA geschriebenen Codes unserer Arbeit ist ausführlich als JAVAdoc vorhanden und dokumentiert. Die Konfiguration der Laufzeitumgebung (Netzwerk und Webserver) und die Implementierung der DB kann in dieser Arbeit als vollständig dargestellt angesehen werden.

Aufbau der Ausarbeitung

Diese Ausarbeitung gliedert sich in drei große Teile.

An erster Stelle befinden sich die Grundlagen die in Abschnitt 2 auf Seite 6 zu finden sind. Diese werden deshalb zu Beginn beschrieben da diese für das Design und die Implementierung des neuen Systems nötig waren. Im Wesentlichen handelt es sich hierbei um Dinge, die für die Einrichtung einer einwandfreie Entwicklungsumgebung, sowie um Werkzeuge die von uns während des Projekts eingesetzt und benötigt wurden. Außerdem sollen grundlegende Einblicke in die JSF und DB-Entwicklung gegeben werden.

Der zweite Teil beschäftigt sich mit der eigentlichen Modellierung (in Abschnitt 3 auf Seite 33) des Systems. Hier sollen die eingesetzten Verfahren der Softwareentwicklung im Bezug auf das zu entwerfende System aufgezeigt werden.

Dem dritten Teil liegt die Implementierung (in Abschnitt 4 auf Seite 58) zugrunde. In diesem Teil soll die Implementierung und Inbetriebnahme der wichtigsten Entwurfsmuster des Projekts, die während der Phase der Modellierung entstanden sind, behandelt werden.

Das methodische Konzept dieser Ausarbeitung ist immer von einfachen zu komplexen und von bekannten zu unbekannten Inhalten gegliedert. Dem Leser werden immer wieder neue Problematiken aufgezeigt und anhand von bekannten Lösungsansätzen anschaulich erklärt. Auch im Aufbau der Arbeit soll ein 'roter Faden' zu erkennen sein.

Konventionen in dieser Ausarbeitung

Aus Gründen der Übersichtlichkeit soll kurz und knapp beschrieben werden welche Bedeutungen andere Formatierungen haben und an welcher Stelle Verzeichnisse zu finden sind.

- Fremdwörter oder Fachbegriffe:

  sind im Text blau geschrieben und besitzen eine Verlinkung zum Glossar im Anhang, in welchem die Begrifflichkeiten indiziert und näher erklärt werden.

- Referenzen von Abbildungen, Listings und Tabellen:

  sind ebenfalls blau und in das jeweilige Verzeichnis im Anhnag verlinkt.

- Abkürzungen (Akronyme):

  stehen in Klammern hinter den dazugehörigen Wörtern. Sie sind darüber hinaus im Abkürzungsverzeichnis aufgelistet.

- `Nichtproportionalschrift`:

  stehen für einzelne Befehle außerhalb von Listings oder für Eigennamen (bspw. bei Softwarepaketen etc...).

- Internet-Links (URLs):

  sind rot und entweder direkt oder als Fußnoten angegeben.

- Anhang:

  enthält zusätzliche Informationen zur Ausarbeitung die aus Gründen der Übersichtlichkeit im Text nicht integriert wurden.

- Literaturverzeichnis:

  ist nach dem Anhang am Ende der Ausarbeitung aufgeführt. Zum Kenntlichmachen von Quellen wurde das Sigle-Verfahren benutzt.

2 Grundlagen

Wie schon zu Beginn erwähnt, sollen in diesem Abschnitt der Ausarbeitung die Grundlagen der von uns verwendeten Technologien und Konzepte angesprochen und näher erläutert werden. Zu Beginn werden die eingesetzten Werkzeuge und Konfiguration der Entwicklungsumgebung besprochen. Anschließend sollen grundlegende Kenntnisse der Webentwicklung mit JSF und PostgreSQL vermittelt werden.

2.1 Eclipse und Enwticklungsunterstützung

Umfangreiche Softwareprojekte lassen sich durch die Verwendung einer Integrated Development Environment (IDE) wie Eclipse oder NetBeans besser bearbeiten. Eclipse wurde ursprünglich von IBM entwickelt und im Jahr 2001 quelloffen veröffentlicht. Im Jahre 2004 übernahm die Eclipse-Foundation die Weiterentwicklung des Projekts und ist heute neben der glsApache Software Foundation eines der größten Open-Source-Communities im Bereich der JAVA-Softwareentwicklung. Seit dem Jahr 2006 wurden verschiedene Versionen und Unter-Projekte von Eclipse auf einen gemeinsamen und aktuellen Stand gebracht. Nach vielen Jahren der Entwicklung und stetigen Verbesserungen ist Eclipse heutzutage eines der beliebtesten IDEs im Bereich der Anwendungsentwicklung. Heute ist Eclipse weit entfernt davon eine IDE nur für JAVA zu sein. Diverse Erweiterungen machten es zum dem Entwicklungstool für Software jedglicher Art und beinahe jeglicher Programmiersprache, schlechthin.

Für die Entwicklung des Projekts selbst wurde Eclipse, Version 4.2 (Juno) verwendet. Außer den Basisfunktionen stehen Entwicklern viele Möglichkeiten offen diverse Plug-ins und Tools in Eclipse zu integrieren, die die Arbeit erleichtern.

Die für dieses Projekt verwendeten Softwareunterstützungen, waren:

- Apache Maven
- Eclipse **WTP!** (**WTP!**)
- JBoss-Tools
- Git-Repository

Juno selbst stellt standardmäsig bereits über die Enterprise Edition, das Projekt WTP, eine Unterstützung für die Entwicklung von JSF bereit. Eine weiteres Tool, die JBoss Inc.-Tools, welche auch eine Erweiterung der Entwicklungsunterstützung im aktuellen Projekt darstellen, werden im übernächsten Unterabschnitt noch genauer erklärt.

Zur Automatisierung einzelner Schritte in der Softwareentwicklung wurde das Build- und Management-Tool Apache Maven zur Umsetzung verwendet.

Desweiteren kam die freie Software zu Versionsverwaltung Git zum Einsatz. Software dieser Art bietet sich vorallem bei kollaborativen Projekten, also Projekten mit mehreren Entwicklern an. Aber auch der Vorteil der Versionskontrolle bei kleinen 'Ein-Mann-Projekten' soll besser nicht unterschätzt werden.

Da es sich hier um selbstständige Tools handelt, die nicht zwingend in einer IDE integriert sein müssen, werden diese jeweils in einem eigenen Unterabschnitt genauer erklärt. Im folgenden soll zuerst auf Maven eingegangen werden.

2.2 Apache Maven

Der sinnvollste Weg ein JSF Projekt zu verwalten ist Apache Maven. Das Tool wird von der Apache Software Foundation entwickelt und dient zur Verwaltung von Softwareprojekten überwiegend im JAVA Umfeld.

Für die Entwicklung und Integration von Maven in Eclipse bietet sich die Installation des Plugins `m2e` bzw. `m2eclipse` an. Die Integration und Verwendung von Maven in Eclipse ist unter [ecl13] näher beschrieben.

Bei Maven steht vorallem die Integration der Webapp sowie die Bibliotheks- und Versionsverwaltung im Vordergrund. Die genaue Funktionsweise wird im späteren Verlauf noch erklärt. Das erstellte Apache Maven-Projekt kann somit in Eclipse importiert und dort bearbeitet werden oder wird schon zu Beginn in Eclipse angelegt. Eine Integration von Apache Tomcat in Eclipse bietet dann die Möglichkeit die Webapp ohne umständliches Erstellen und Installieren auf dem Servlet-Container zu verwenden.

In Abbildung 1 auf Seite 8 wird die Verwaltung mit Hilfe von Maven darstellt.

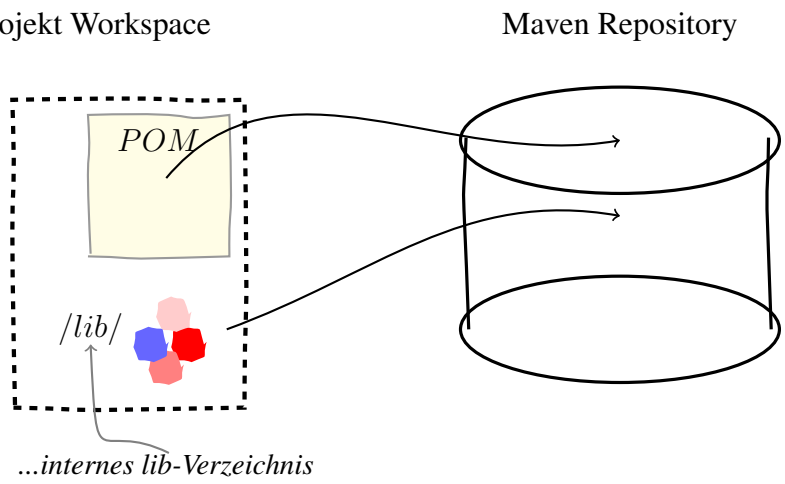


Abbildung 1: Skizze der Maven Verwaltung

Ein grundlegendes Konzept von Maven lautet: 'convention over configuration' (Konvention vor Konfiguration). Konkret bedeutet dies, dass sich Maven standarmäßig immer so verhält, wie es für die meisten Projekte sinnvoll ist und nur bei Abweichung von Normen konfiguriert werden soll. Zu diesem Konzept gehört auch die Verzeichnisstruktur, die standardmäßig verwendet wird, bei Bedarf aber angepasst werden kann.

Der Aufbau einer Verzeichnisstruktur für Webandwendungen ist folgendermaßen aufgebaut:

Maven bildet die Quellen im Verzeichnis `src` und die lauffähige Version des Programms als `*.war` im Verzeichnis `target` ab. Dies beschreibt den Fall einer Standarkonfiguration für Apache Tomcat-Webapplikationen. Apache Maven-Projekte können aber auch anhand von Archetypen erstellt werden. Dieses Projekt basiert auf einem 'Custom Project' welches JSF-myFaces Core 2.0 nutzt.

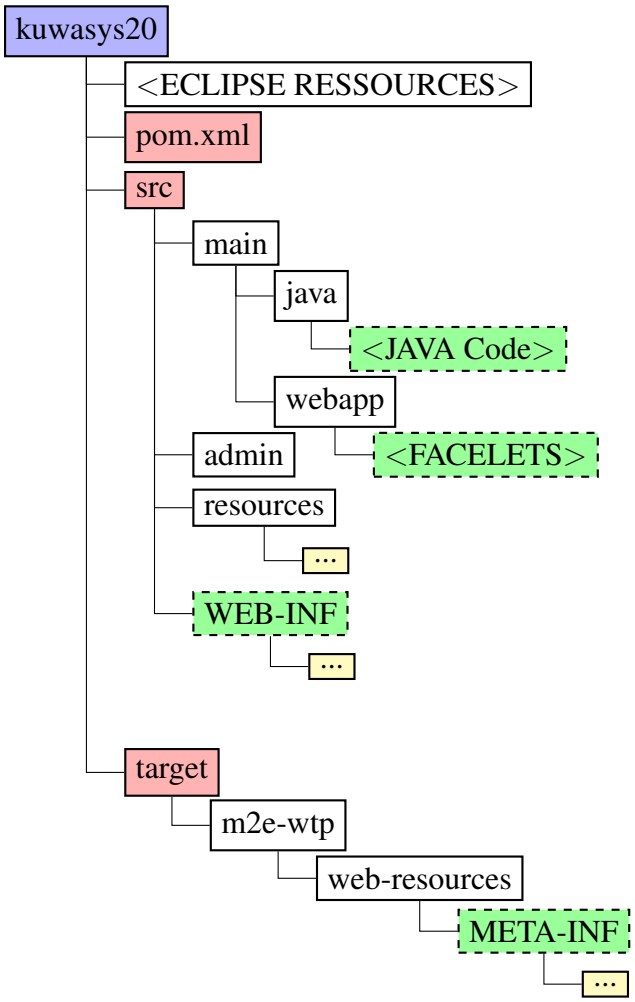


Abbildung 2: Maven Projekt Verzeichnisstruktur

Das Herzstück eines Maven-Projekts bildet das Project Object Model (POM), welches durch die Datei `pom.xml` repräsentiert wird. Ein Ausschnitt ist in Quellcodeausschnitt 1 zu sehen. Darunter finden sich unter anderem Informationen über die Art des Projekts, Angaben zur Art des Erstellvorgangs und Abhängigkeiten wie z.B. der JSF-myFaces Core-API, JSF-myFaces Tomahawk, und PostgreSQL.

Mit dem Element `packaging` wird angegeben, dass das Projekt als WAR-

Archiv gepackt werden soll.

Das Element `dependencies` enthält alle Softwareabhängigkeiten. Für jede Abhängigkeit wird ein `dependency`-Element angegeben. Es wird auch Artefakt genannt, welches mit `groupID`, `artifactID` und einer Versionsnummer beschrieben wird, welches in Kombination einzigartig ist. Das Element `scope` gibt an, in welchem Classpath (Laufzeit, Kompilierzeit oder Ausführung von Tests) ein solches Artefakt verfügbar sein soll. Wichtige Werte für das Element sind:

`compile`,

standardmäßig definiert. Es bedeutet, dass ein Artefakt in jedem Classpath vorhanden sein soll.

`provided`,

Artefakte sind nur beim Kompilieren im Classpath enthalten und werden nicht in die fertige Anwendung integriert. Das ist dann sinnvoll wenn Artefakte zum Kompilieren benötigt werden, aber beim Ausführen sich ohnehin im JDK befinden.

`runtime`,

die Artefakte werden zum Ausführen und Testen benötigt, aber nicht zum Kompilieren. Sind sind in der fertigen Anwendung enthalten.

`test`,

wenn Artefakte fürn Tests beötigt werden. Sie sind nicht in der fertigen Anwendung enthalten und außerdem nur beim Kompilieren und Ausführen von Tests verfügbar. Alle hier aufgeführten `scopes` sind in [mav13] zu finden.

```
<packaging>war</packaging>
<dependencies>
  <dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-api</artifactId>
    <version>2.1.10</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-impl</artifactId>
    <version>2.1.10</version>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

Code 1: Ausschnitt der `pom.xml`

Beim Auflösen der Abhängigkeiten eines Projekts, bspw. während dem Kompiliervorgang, prüft Maven zunächt an welcher Stelle sich benötigte Artefakte befinden. Zunächst wird das lokale Repository geprüft. Hierbei handelt es sich um ein Verzeichnis im lokalen Dateisystem. Ist das nicht der Fall, versucht Maven sich die Artefakte über das Central Repository, einer durch Apache bereitgestellten Sammlung von Artefakten, herunterlzuladen.

2.3 Verteilungs- und Versionskontrolle mit Git

Wie bereits erwähnt wurde das Projekt mit der Hilfe der Versionskontrolle Git realisiert. Git stellt, neben Subversion (SVN) und Mercurial, das beliebteste Werkzeug zur Verteilung von Daten mit Versionskontrolle dar.

Abbildung 3: Git Logo

Entwickelt wurde Git ursprünglich für Aktualisierungen des Linux-Kernels. Es verbreitete sich jedoch schnell in Kreisen der Softwareentwicklung für Verteilung, Versionskontrolle und Backup. Software dieser Art wird dazu verwendet alle Dateien eines Projekts bei jedem Mitwirkendem aktuell und gleich zu halten. Ein weiterer Vorteil ist es hierbei, dass auf jede beliebige Vorgängerversion zurückgegriffen werden kann. So kann bei schwerwiegenden Fehlern zu einer funktionstüchigen Version gesprungen werden. Der positive Nebeneffekt sind daher auch Backups des kompletten Projekts die im sogenannten Git-Repository sind.

Obwohl der Sinn und Zweck im allgemeinen der gleiche ist, trifft man bei Git auf einige Unterschiede gegenüber traditionelleren Versionskontrollsystemen wie bspw. SVN, die kurz beschrieben werden sollen:

Die Verteilung der Daten wird nicht wie bei SVN über eine zentrale Client-Server Struktur realisiert sondern über eine dezentrale Client-Server Struktur, ähnlich wie bei Peer-to-Peer (P2P) Netzwerken ([PM07], 57). beschrieben sind. Die Abbildung 4 stellt so eine Netzwerkverteilung dar. Dabei sind die farblichen Unterschiede der Kanten als eine Verbindung zu einem Git-Repository zu interpretieren. Die gestrichelten Linien stellen noch nicht etablierte Verbindungen dar.

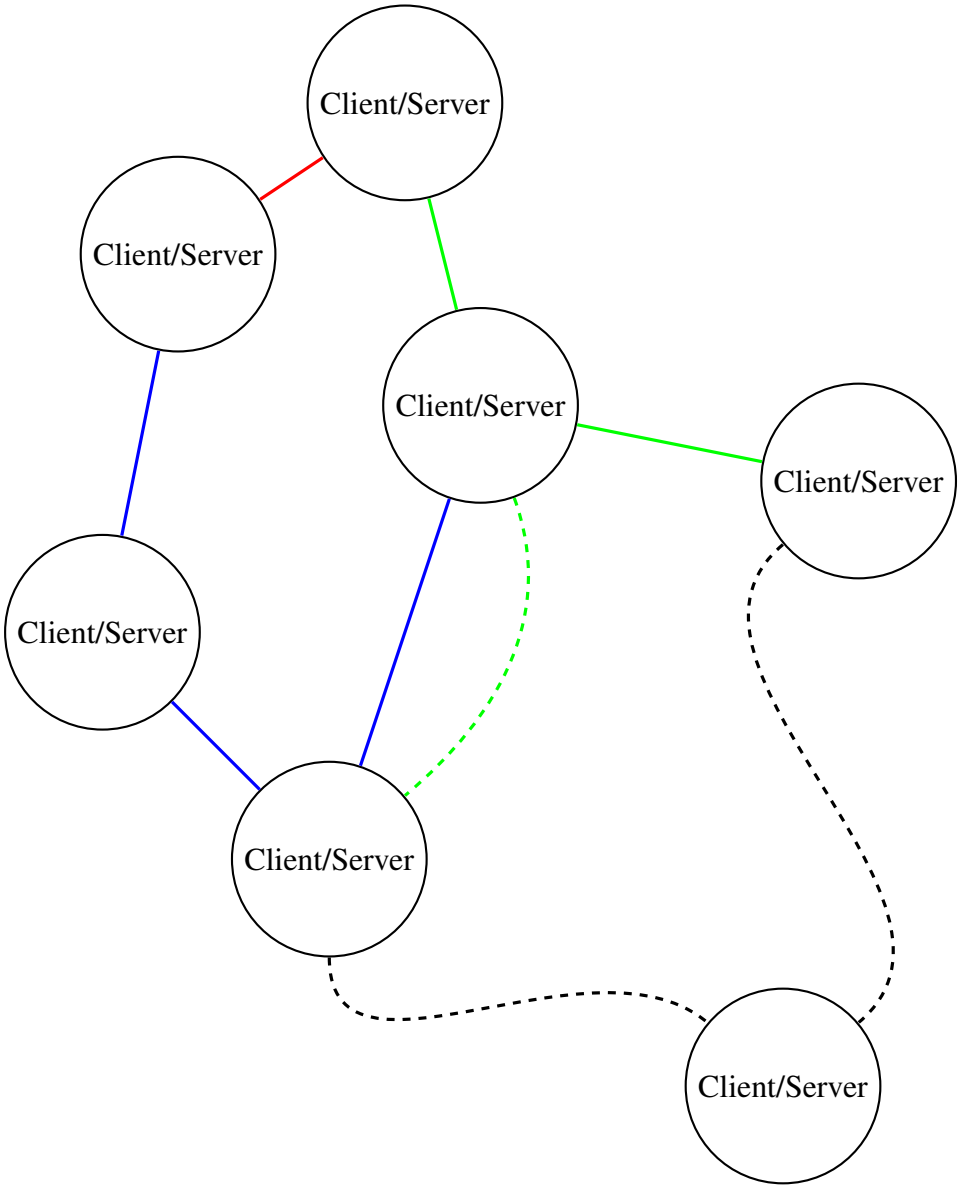


Abbildung 4: Git-Netzwerkstruktur

Für die Git-Verwaltung des Projekts, wurde Github [2] verwendet. Um mit Github arbeiten zu können wird ein gültiger Account vorrausgesetzt. Mit Git kann in der Konsole gearbeitet werden oder es existieren Plugins für die IDE, was die Arbeit und Entwicklung weniger aufwendig macht.

[2] https://github.com/

An dieser Stelle soll kurz gezeigt werden wie mit Git über die Konsole gearbeitet werden kann.

Ist bereits ein Git-Repositoty angelegt kann wie folgt vorgegangen werden:

1. das bereits existierende Verzeichnis mit dessen gesamten Inhalt wird 'gecloned'. Danach können beliebige Veränderungen vorgenommen werden

```
1 git clone https://github.com/[name]/[repository.git]
```

Code 2: Git: git clone

2. sollen die Änderungen wieder übernommen werden, müssen die entsprechenden Dateien hierfür übernommen werden. In diesem Fall werden durch den Parameter * alle Dateien im Verzeichnis hinzugefügt

```
1 git add *
```

Code 3: Git: git clone

3. Anschließend müssen die Dateien für das Aktualisieren vorbereitet werden. Eine Nachricht ist hierzu notwendig und wird durch den Parameter -m angegeben

```
1 git commit -m [message]
```

Code 4: Git: git clone

4. schließlich müssen wir die Dateien ans Repository übertragen.

```
1 git push origin master
```

Code 5: Git: git clone

Ist bisher, noch kein Repository eingerichtet und ein neues soll erstellt werden, läuft der Vorgang wie folgt ab:

1. Git initialisieren. In diesem Fall wird das aktuelle Verzeichnis also Git-Repository deklariert. In diesem kann später auch gearbeitet werden

```
git init
```

Code 6: Git: git clone

2. Dateien die übertragen werden sollen werden also dem Paket hinzugefügt (`add`)
3. die hinzugefügten Dateien werden für das übertragen vorbereitet (`commit`)
4. nun muss ein Git-Repository (in diesem Fall auf Github) angelegt werden. In diesem Beispiel wird die Verbindung über HTTPS aufgebaut. Git unterstützt aber ebenso auch Verbindungen über SSH. Diese sind generell aber mit etwas mehr Aufwand zu realisieren.

```
git remote add origin https://github.com/[username]/[
   verzeichnisname].git
```

Code 7: Git: git clone

5. zuletzt können die vorgesehenen Daten dem Repository hinzugefügt (`push`) werden

Allgemeine Informationen zu Git sind auf der Git-Website [3] zu finden. Detailierte Hilfestellungen und die genaue Bedienungsanleitung kann in der offiziellen Git-Dokumentation [scm13] nachgeschlagen werden.

2.3.1 JBoss Tools

Ein weiteres praktisches Tool zur Entwicklungsunterstützung mit JSF ist das JBoss-Tools Paket, welches über den Download im Eclipse-Marketplace [4] direkt bezogen

[3] http://git-scm.com/
[4] http://marketplace.eclipse.org/node/420896#.UZS9yKwaSKI

werden kann. Während der Programmierung einzelner Facelets rendert das Tools die dazugehörige Ansicht und stellt es dem Entwickler in Echtzeit dar.

Das verschafft Vorteile beim Einstieg in Java Server Faces, aber auch dann, wenn bei der Implementierung schnell getestet werden soll.

2.4 Java Server Faces

JSF ist eine auf JAVA basierende Webtechnologie die auf den Standards der Servlets- und JSP-Techniken aufsetzt. Es ist ein eigenes plattformunabhängiges Framework zum vereinfachten Erstellen der Benutzeroberflächen in Webapps. Dies wird dadurch erreicht, dass der Entwickler die Möglichkeit hat die Benutzerschnittstellen auf eine einfache Art und Weise in eine Website einzubinden und die gesamte Navigation serverseitig zu definieren. Darüber hinaus können alle clientseitigen Events an serverseitige Handler gebunden werden.

Voraussetzungen zur Entwicklung mit JSF sind Grundkenntnisse mit der Programmiersprache JAVA und mit dem damit verbundenen JDK sowie dem HTTP Protokoll. Zur Darstellung wird ein Servlet-Container, zum Beispiel ein Apache Tomcat, benötigt. Für die Obeflächengestaltung der Webiste ist ein grundlegendes Verständnis der HTML-Technik von Vorteil.

Zunächst soll in diesem Abschnitt der Arbeit die grundlgende Funktionsweise des JSF-Standards dargestellt und beschrieben werden.

2.4.1 Das JSF-Prinzip

Die zentrale Idee von JSF beruht auf dem Konzept der Komponenten. Diese fördern die Wiederverwendbarkeit des UI Codes in anderen Projekten und verhindern unnötige Coderverdopplung.

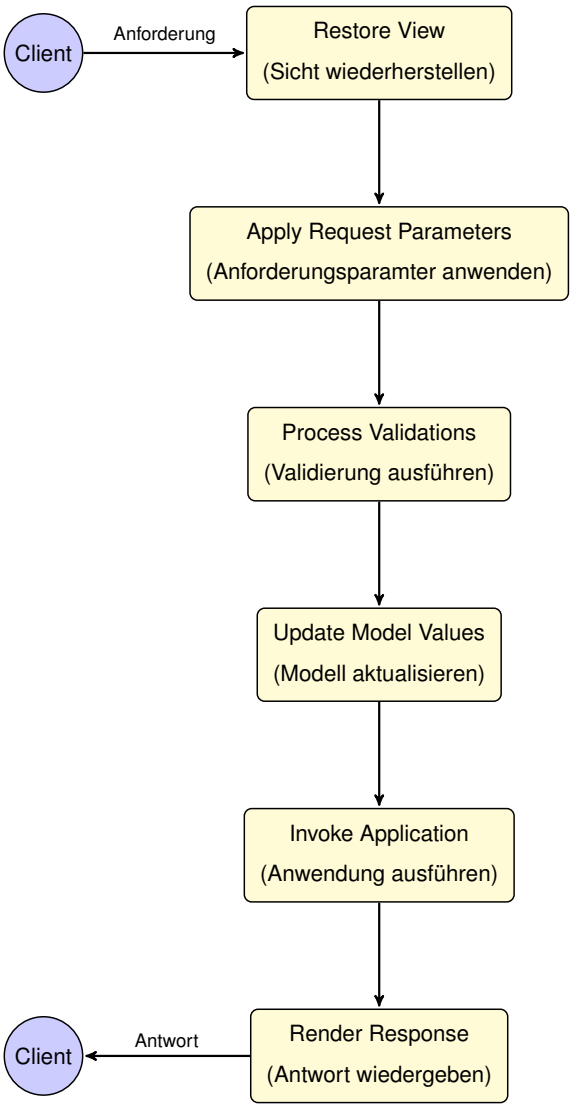


Abbildung 5: Diagramm des Lebenszyklus in JSF

Um eine einheitliche Struktur der Anwendung zu schaffen wird nach dem Model-

View-Controller (MVC) Prinzip das Modell, die Präsentation und die Steuerung voneinander getrennt, verdeutlicht wird dies unter [jsf13b]. Auch die Abbildung 6 auf Seite 18 erläutert das Modell im Bezug auf den JSF Lebenszyklus detailiert.

Unterstützt wird dieses Konzept durch die sogenannte View-Komponente, welche als Baumstruktur der JSF-Komponenten interpretiert werden können. Jede JSF-Anwendung unterliegt einem Lebenszyklus welcher auf der Abbildung 5 auf Seite 17 zu sehen ist, der View steht zu Beginn eines Zyklus.

Vor dem Ende des Zyklus wird das Wurzelelemnt des Views erneut rekursiv aufgerufen. Dieser Vorgang generiert die Antwort, welche bspw. eine HTML-Seite sein kann. Für die Deklaration der Webansicht gibt es viele Möglichkeiten. Die hier genannten Erklärungen zum Lebenszyklus einer JSF Anwendung sind in [jsf13a] beschrieben. Betrachtet man bspw. andere Frameworks wie Turbine so dient als Seitendeklarationssprache, englisch View Declaration Language (VDL) genannt, Velocity und bei Cocoon ein abstrahierter XML-Dialekt.

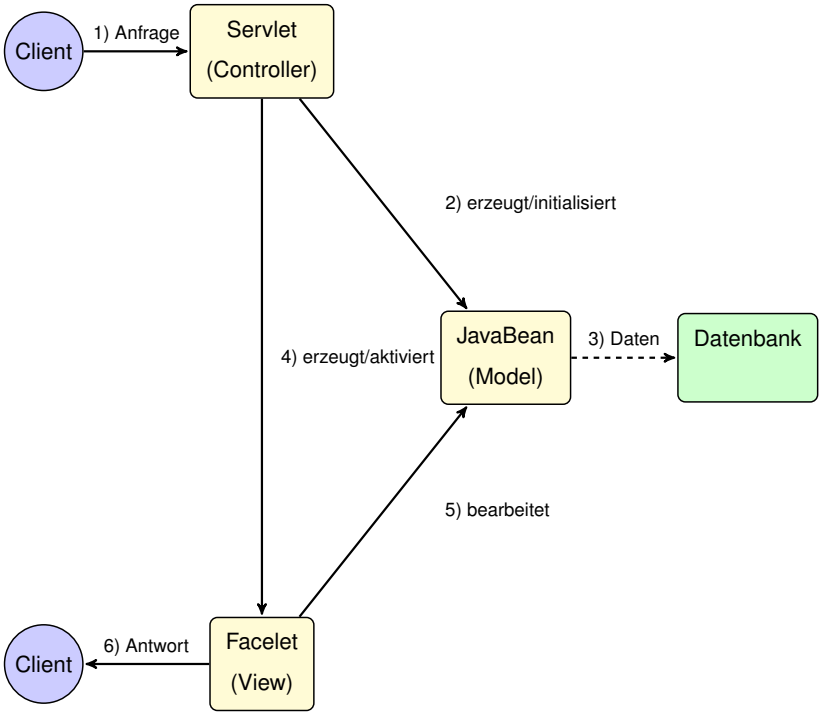


Abbildung 6: Model-View-Controller Modell in JSF

Bei JSF (und im übrigen auch bei Struts) vor Version 2.0 wurden Java Server Pages eingesetzt. Erst ab der Version 2.0 setzt JSF standardmäßig XHTML, bekannt als die sog. Facelets auf welche in Unterabschnitt 2.4.2 auf Seite 19 näher eingegangen wird, als Seitendeklarationssprache ein. XHTML entwickelte sich aus HTML und XML, nachdem sich gezeigt hatte, dass HTML selbst nicht XML konform ist. XHTML wird als Schnittmenge aus HTML und XML verstanden ([AB03], 33).

Benutzereingaben können über eigene Komponenten oder mit einem Data Handler, aus JAVA Code bestehend, geschrieben werden. Diese Controller-Komponente entspricht dem Controller im MVC Modell. Die letzte Komponente, die Model-Komponente, beinhaltet die Logik der Webapp. In JSF sind diese Komponenten die sog. Java-Beans, welche vom Servlet-Container verwaltet werden.

Die eigentliche Ansicht der Website, die dem Betrachter zur Verfügung gestellt wird, wird durch generierten HTML Code der Seite sichtbar, dies geschieht durch einen eigenen JSF-Renderer. Darüber hinaus bietet JSF den Entwicklern weitere Möglichkeiten, sich eigene Renderer zusammenzustellen ([de.13b]).

2.4.2 Darstellung von Seiteninhalten mit Facelets

Bisher wurden lediglich Eindrücke die auf die Konzeption und die allgemeine Verarbeitung von Informationen in JSF eingehen, vermittelt. Nun soll die Darstellung der Informationen durch JSF erklärt werden.

Die JSF-Technologie unterstützt, wie schon im vorherigen Abschnitt erwähnt, mehrere Arten zur Anzeige von Seiteninhalten. Vor Version JSF 2.0 wurde standarmäßig JSP als Seitendeklarationssprache eingesetzt um die neu entwickelte Technologie zu etablieren und um für Entwickler die Probleme der Sprachbarrieren niedrig zu halten. Da JSP noch immer ein weit verbreiteter Standard ist, wird es auch noch heute für JSF

Projekte eingesetzt. Das große Problem bei dieser Art der Implementierung ist, dass die Kombination von JSF und JSP keine gute Lösung darstellt. Die Problematik rührt daher, dass beide Technologien für unterschiedliche Einsatzzwecke entworfen wurden.

Wie oben schon beschrieben wird eine JSF Applikation in mehreren Phase abgearbeitet. Der Aufbau der Struktur von Komponenten und die Darstellung der Ausgabe wird hierbei in mehreren Phasen realisiert. Im Gegensatz hierzu: JSP, welches mit nur einer einzigen Phase abgearbeitet wird. Die Antwort vom Server wird direkt in ihr ausgegeben. JSP Seiten stechen vor allem durch ihren Aufbau hervor. Die Ausgabe die der Sever darstellen muss wird in einer JSP-Seite immer durch die Verwendung der JAVA-Funktion `out.println()` in HTML Seiten generiert, was zur Folge haben kann, dass Seiteninhalte vor anderen dargestellt werden oder das Design an Konsistenz verliert. Das geschieht vor allem dann wenn normale HTML- und Text-Elemente zum Einsatz kommen.

Um diese Problematiken in den Griff zu bekommen wurden die Facelets entwickelt. Hierbei handelt es sich ebenfalls um eine VDL, allerdings ist die dem Lebenszyklus von JSF-Applikationen angepasst. Hierzu werden Inhalte aus XHTML-Dokumenten wie sie in Abbildung **??** auf Seite **??** zu sehen ist verknüpft um einen strukturierten Komponentenbaum aufzubauen.

Code 8: Hierarchische Struktur der einzelnen Facelet-Elemente

```
1 <?xml version="1.0"?>
2 <ui:composition template="template.xhtml">
3   <ui:define name="title">Baumstruktur</ui:define>
4   <ui:define name="content">
5     <h3>Inhalt</h3>
6     <p>Das ist ein Test-Text</p>
7   </ui:define>
8 </ui:composition>
```

Wie oben schon erwähnt sind Facelets mittlerweile in JSF 2.0 zum Standard geworden, JSP wird nur noch aus Gründen der Kompatibilität unterstützt und besondere Features lassen sich in JSF sogar nur noch mit Hilfe von Facelets umsetzen.

Aufgrund der Tatsache, dass Facelets speziell für JSF konzipiert und entwickelt wurden, bieten sie natürlich viele Vorteile. Das Erstellen und Ausgeben von Seitenansichten wird mit Facelets effizienter. Dies schlägt sich in der Entwicklung von Seiten so wie in deren Geschwindigkeit beim Seitenaufbau nieder.

Zur Unterstützung der Wiederverwendbarkeit können JSF Seiten mit Hilfe der Facelets Template-basierend aufgebaut werden. Das bedeutet, dass komplette Seitenfragmente zentral abgelegt werden können und bei Bedarf in andere Seiten eingebettet werden. So kann bspw. ein Layout für einen Seitenkopf einmal definiert werden, um ihn dann später in anderen Seiten wiederverwenden zu können. Dies verbessert zum einen die Wartbarkeit der Seiten und birgt den Vorteil der Modularisierung in ganzen Projekten.

Im Fall von Quellcodeausschnitt 9 auf Seite 21, welcher ein Beispiel des JSF-Templatings minimalistisch darstellt, wird ein HTML-Gerüst erzeugt. Es können zusätzlich globale CSS- und JavaScript-Dateien eingebunden werden. Die Elemente `ui:insert` definieren Platzhalter für beliebige Daten, die an dieser Stelle eingefügt werden können. Der Quellcodeausschnitt 8 auf Seite 20 zeigt eine Seite, die das gerade betrachtete Template verwendet. Hierzu wurd im Element `ui:composition` das gewünschte Template zur Verwendung festgelegt. Nun wird mit `ui:define`, dessen `name`-Attribut dem `name`-Attribut des Elements `ui:insert` aus dem Template entsprechen muss, der Inhalt festgelegt.

Code 9: Minimalbeispiel für ein Template

```
1 <h:head>
2         <title>Kurswahlsystem der Schillerschule Aalen - <ui:
             insert name="title" /></title>
3         <ui:insert name="headers" />
```

```
4  </h:head>
5  <h:body>
6  <h:outputStylesheet library="css" name="style.css"/>
7          <t:div id="container">
8                  <h:graphicImage library="img" name="header.jpg
                      " alt="HeaderImage"/>
9                  <t:div id="titlebar">
10                         <h:panelGrid columns="3" style="text-
                              align:right; width:980px;">
11                                 <ui:insert name="title" />
12                                 <h:outputText style="color:
                                      lightgray;" rendered="#{
                                      kuwasys.userRole()==null?
                                      false:true}" value="
                                      Schuljahr #{kuwasys.year
                                      }/#{kuwasys.year+1} |
                                      Tertial: #{kuwasys.tertial}
                                       | #{kuwasys.systemPhase()}
                                      "/>
13                                 <h:outputText style="color:
                                      lightgray;" rendered="#{
                                      kuwasys.userRole()==null?
                                      false:true}" value="#{
                                      userBean.showUsername()} |
                                      #{request.remoteUser}
                                      &#8834; #{kuwasys.userRole
                                      ()}"/>
14                         </h:panelGrid>
15                 </t:div>
16                 <h:panelGrid columns="2">
17                 <ui:include src="/navigation.xhtml"/>
```

```
18                <t:div id="content"><ui:insert name="content"
                     /></t:div>
19                </h:panelGrid>
20          </t:div>
21          <t:div id="footer">Kurswahlsystem - Schillerschule
               Aalen - MI-Projektarbeit im WS12/13 HTW-Aalen -
               Christian Silfang/Michael Sch tz </t:div>
22  </h:body>
```

Wird der definierte Quellcodeausschnitt 9 auf Seite 21 aufgerufen, so würde folgende HTML-Seite erzeugt werden:

Code 10: Gerenderte HTML-Seite des selbstdefinierten Templates

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Baumstruktur</title>
5    </head>
6    <body>
7      <h3>Inhalt</h3>
8      <p>Das ist ein Test-Text</p>
9    </body>
10 </html>
```

Neben den Templates können auch komplett eigene Komponenten erstellt werden. Das ist vor allem dann vorteilhaft wenn HTML-Code nach eigenen Vorstellungen eingebettet werden soll. Hierzu wird über das `jsfc`-Attribut der Facelet-Compiler angewiesen beliebige XML-Elemente in Komponenten umzuwandeln. Ein Beispiel zeigt der Quellcodeausschnitt 11 auf Seite 23

Code 11: Beispiel einer selbst erstellten Input-Komponente

```
1 <input id="vorname" jsfc="h:inputText"
2     value="#{userBean.vorname}"/>
```

Im nächsten Abschnitt soll nun das dynamische erzeugen von Seiten mit JSF erklärt werden.

2.4.3 Managed-Beans in JSF

In der Hierachie unter den Komponenten liegen die Managed-Beans (teilweise auch Backing-Beans genannt). Sie sind dafür verantwortlich, dass Werte zum 'Befüllen' von Komponenten geliefert werden. Diese Werte können Initialwerte sein, welche Managed Properties genannt werden.

Managed-Beans sind zentral definierte simple JAVA-Klassen, sogenannte POJOs, die dem Java-Beans Standard entsprechen. Je nach Gültigkeitsbereich, auch Scope genannt, der für die Managed-Beans deklariert ist, können diese für die gesamte Applikation oder nur für einzelne Benutzer gültig sein. Deklariert werden können sie in der XML Konfiguration des JSF Projekts. Ab JSF 2.0 geht dies auch über Annotations über der Klassendefinition direkt im JAVA-Code.

In JSF kann also auf die Eigenschaften der Managed-Beans, lesend und schreibend, zugegriffen werden. Eine solche Eigenschaft besteht aus einem Namen, einem Typ und `set`- und `get`-Methoden zum Lesen und Schreiben der Werte. Die Namen solcher Methoden unterliegen gewissen Konventionen:

- `get`[Eigenschaftsname]: lesender Zugriff
- `set`[Eigenschaftsname]: schreibender Zugriff

Der Quellcodeausschnitt 12 auf Seite 25 zeigt eine Managed-Bean Klasse mit jeweils 2 Methoden für einen der beiden privaten Strings der Klasse, `firstName` und `lastName`.

Code 12: UserBean mit get- und set-Methoden

```
1  import javax.faces.bean.ManagedBean;
2  import javax.faces.bean.SessionScoped;
3
4  @ManagedBean
5  @SessionScoped
6  public class Customer {
7    private String firstName;
8    private String lastName;
9
10   public String getFirstName() {
11     return firstName;
12   }
13   public void setFirstName(String firstName) {
14     this.firstName = firstName;
15   }
16   public String getLastName() {
17     return lastName;
18   }
19   public void setLastName(String lastName) {
20     this.lastName = lastName;
21   }
22
23   public String sendUser() {
24     // FORMULAR-LOGIK
25     return "userAdd";
26   }
```

```
27 }
```

Dabei ist egal ob mit den Methoden auf private Variablen zugegriffen wird oder ob komplexere Algorithmen ausgeführt werden, es ist in jedem Fall nur, die Eigenschaft auf die zugegriffen wird, sichtbar.

Das dazugehörige Facelet stellt der Quellcodeausschnitt 13 auf Seite 26 dar. Das Facelet zeigt ein kleines Eingabeformular, in welches einmal der Vorname und einmal der Nachname über Eingabefelder vom User eingegeben werden können. Im Facelet wird dies durch die Angabe der zu verwendenden Property `#{userBean.firstName}` und `#{userBean.lastName}` realisiert. Die Werte die im Textfeld eingegeben werden, werden auch von der Managed-Bean verarbeitet, indem die Setter-Methode, durch den Audruck des Attributs `value`, für das Feld `firstName` bzw. `lastName` der Bean `Customer` aufruft und die eingegebenen Werte weiter verarbeitet.

So kann also eine wichtige Aufgabe mit JSF relativ unproblematisch erledigt werden. Nun sollen die Daten die ins Formular eingegeben werden weiterverarbeitet werden. Hierzu wird im Formular der Button mit `action=ßendUser()"` definiert. Beim Klick wird die Aktion `sendUser()` in der dazugehörigen Bean aufgerufen und die Daten aus dem Formular können wie gewünscht weiterverarbeitet werden.

Gleiches Prinzip funktioniert ebenfalls für eine Ausgabe am Bildschirm, bspw. über den xhtml-Tag `h:outputText`. Es würde die relevante Getter-Methode aufgerufen werden und wiederum das dazugehörige Feld ausgegeben werden.

Code 13: UserAdd Facelet zur UserBean mit Eingabeformular

```
1 <?xml version="1.0"?>
2 <ui:composition template="template.xhtml"
3   xmlns:f="http://java.sun.com/jsf/core"
4   xmlns:h="http://java.sun.com/jsf/html"
```

```
5   xmlns:ui="http://java.sun.com/jsf/facelets">
6   <ui:define name="title">Test</ui:define>
7   <ui:define name="content">
8     <h1>Vornamen und Nachnamen eingeben</h1>
9     <h:form id="studentadd">
10      <h:panelGrid columns="1">
11        <h:outputLabel for="name" value="Vorname␣eingeben:" />
12        <h:inputText id="name" value="#{userBean.firstName}"
             required="true" />
13
14        <h:outputLabel for="lastname" value="Nachname␣
             eingeben:" />
15        <h:inputText id="lastname" value="#{userBean.lastName}
             " required="true" >
16      </h:panelGrid>
17      <h:commandButton value="Schueler␣anlegen" action="#{
           userBean.sendUser}" />
18    </h:form>
19  </ui:define>
20 </ui:composition>
```

Ein Ausdruck beginnt immer mit einem den Zeichen `$` oder `#`. Begrenzt wird der Ausdruck durch geschweifte Klammern. Der Unterschied des `$`-Zeichens zum `#`-Zeichen ist, dass mit ihm nur lesend auf ein Property zugegriffen werden kann. Der Ausdruck wird sofort evaluiert und ausgegeben. Er eignet sich daher für einfach Textausgaben. Ein Beispiel hierzu ist in Quellcodeausschnitt **??** auf Seite **??** zu betrachten.

Diese Ausdrücke sind Teil der Unified Expression Language (UEL), also eine Art von vordefinierten Facelet-Tags, mit deren Hilfe auf alle Properties einer Managed-Bean zugegriffen werden kann.

Code 14: UserAdd Facelet zur UserBean mit Ausgabe

```
<?xml version="1.0"?>
<ui:composition template="template.xhtml"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <ui:define name="title">Test</ui:define>
  <ui:define name="content">
    <h1>Vornamen und Nachnamen ausgeben</h1>
    <h:form id="studentadd">
      <h:panelGrid columns="1">
        <h:outputLabel for="name" value="Der␣Vorname␣ist:" />
        <h:outputText id="name" value="${userBean.firstName}"
           required="true" />

        <h:outputLabel for="lastname" value="Der␣Nachname␣ist:
           " />
        <h:outputText id="lastname" value="${userBean.lastName
           }" required="true" >
      </h:anelGrid>
      <h:commandButton value="Schueler␣anlegen" action="#{
         userBean.sendUser}" />
    </h:form>
  </ui:define>
</ui:composition>
```

Die nachstehende Tabelle zeigt die möglichen Gültigkeitsbereiche für Managed-Beans:

XML Wert	Annotation	Gültigkeitsbereich
request	@RequestScoped	gültig solange der aktuelle Request behandelt wird
view	@ViewScoped	gültig solange die aktuelle View gültig ist
session	@SessionScoped	gültig für eine Session
application	@ApplicationScoped	gültig für die gesamte Applikation
none	@NoneScoped	gültig solange das Bean gültig ist

Tabelle 1: Gültigkeitsbereiche für Managed-Beans

Der Tabelle kann entnommen werden, dass die verschiedenen Scopes auch unterschiedlich lange Lebensdauern in einer Webapplikation besitzen. Spezialfall ist hier der Gültikeitsbereich `@NoneScoped`. In diesem wird die Managed-Bean nicht gespeichert sondern bei jedem Aufruf neu erstellt. Der `@ApplicationScope`-Gültigkeitsbereich hingegen bildet den Gegensatz. Dieser gilt für die gesamte Lebendauer der App.

2.5 PostreSQL

PostgreSQL ist quelloffen und ein sehr mächtiges objekt-relationales Datenbankmodell und -system. Mittlerweile unterliegt es mehr als 15 Jahren aktiver Entwicklung und wird stetig weiter verbessert. Aufgrund dieser langen Zeit konnte sich

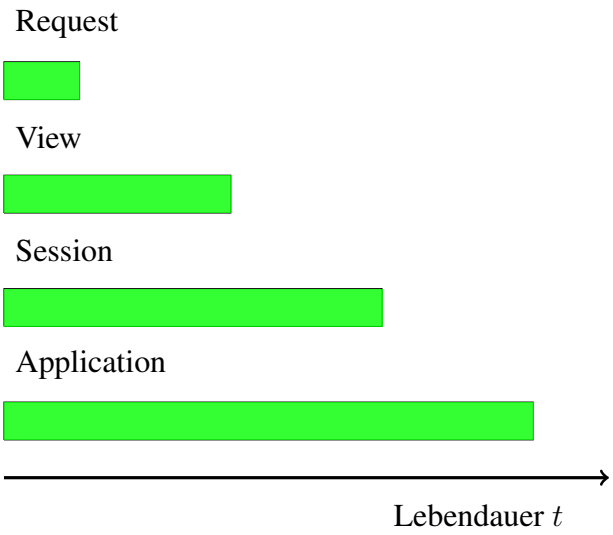


Abbildung 7: Scope Lebensdauern

PostgreSQL einen guten Namen machen und wird aufgrund der Einfachheit Daten zu integrieren und zu verwalten häufig eingesetzt. Ein großer Vorteil von PostgreSQL ist desweiteren, dass es von allen gängigen Betriebssystemen wie Linux, Unix (BSD, Solaris, Mac OS X) und Windows, unterstützt wird. Somit existieren auch grafische Benutzeroberflächen zur DB-Verwaltung, wie das von uns eingesetzte pgAdmin III. Neben den Betriebssystemen werden auch nahezu alle gängigen Programmiersprachen unterstützt als Beispiele seien C/C++, JAVA, .Net, Perl, Python und Ruby genannt. Diese allgemeinen Informationen sowie die Tabelle 2 auf Seite 30, welche die Speicheverwaltungswerte auflistet, wurden der offiziellen PostgreSQL-Projektseite, welche unter [pos13a] zu finden ist, entnommen. Ein Download-Paket des gesamten DB-Systems kann unter [pos13b] bezogen werden.

Limit	**Wert**
Maximale DB Größe	Unbegrenzt
Maximale Tabellen Größe	32 TB
Maximale Zeilen Größe	1.6 TB
Maximale Feld Größe	1 GB
Maximale Anzahl der Zeile pro Tabelle	Unbegrenzt
Maximale Spalten pro Tabelle	250 - 1600 abhängig vom Spaltentyp
Maximale Anzahl an Indizes per Tabelle	Unbegrenzt

Tabelle 2: Speicherverwaltung von PostgreSQL

Da das 'KuWaSys' mit JSF implementiert ist, ist eine JDBC-Schnittstelle unter zu Hilfenahme eines PostgreSQL-Treibers zum Einsatz gekommen. Diese Konzept bietet in Java die einfachste Möglichkeit um auf eine PostgreSQL DB zuzugreifen. Bei dem

verwendeten Treiber handelt es sich um ein JDBC-Treiber des Typ-4. Typ-4 bedeutet, dass die JDBC-API Befehle direkt in Datenbank Managment System (DBMS) Befehle des jeweiligen Datenbankservers übersetzt werden. Es wird kein Middleware-Treiber benötigt. Der Typ-4 ist damit der schnellste (wenn schnelle Netzwerkprotokolle verwendet werden) unter den vier verschiedenen Typen, dafür jedoch aufgrund der fehlenden Middleware weniger flexibel. Dadurch ist ein Treiber des Typ-4 besonders gut für Webapps im Intranet geeignet, weshalb das 'KuWaSys' mit Hilfe des Typ-4 Treibers umgesetzt ist.

Der Quellcodeausschnitt 15 auf Seite 31 zeigt die Einbettung des JDBC-PostgreSQL Treibers in den JAVA Code. Dazu zeigt er einen beispielhaften Verbdindungsaufbau zur Datenbank mit Default-Werten und eine Test-String Ausgabe. Das JDBC-PostreSQL JAR-Paket wurde hierzu bereits in Eclipse integriert. Dies kann über den Einstellungs-Wizard in den Projekt-Eigenschaften bewerkstelligt werden.

Code 15: Herstellung einer SQL-Konektivität in JAVA

```
1  public void SQLConnection() {
2    try {
3      InitialContext cxt = new InitialContext();
4        DataSource ds = (DataSource) cxt
5          .lookup("java:/comp/env/jdbc/postgres");
6        connection = ds.getConnection();
7        System.out.println("DB open");
8        statement = connection.createStatement();
9        result = statement.executeQuery("SELECT VERSION()"); //
             DEBUG
10       if (result.next()) {
11         System.out.println(result.getString(1)); // DEBUG
12       }
13   } catch (SQLException ex) {
14     System.out.println("Error during DB connection " + ex);
```

```
15      ex.printStackTrace();
16    } catch (NamingException ex) {
17      System.out.println("Error during DB connection " + ex);
18      ex.printStackTrace();
19    }
20  }
```

3 Modellierung

Die erste Phase der aktiven Softwareentwicklung befasst sich mit den Fragen zum System. Dabei gilt es besonders darauf zu achten, das vermeintliche 'triviale' Aufgabenstellungen mit berücksichtigt werden und nicht etwa vernachlässigt oder vergessen werden. In Unterabschnitt 3.2 auf Seite 36 wird näher auf die diese Problematik eingegangen. Dieses Problem tritt vor allem dann auf, wenn das System in einer späteren Phase des Projekts noch sinnvoll erweitert werden soll oder in Zukunft eine bestimmte Funktionalität verbessert werden muss.

Um dem Leser einen möglichst genauen Einblick in die Phasen der Modellierung geben zu können, ist der Aufbau dieses Abschnitts wie in Unterabschnitt 1.3 auf Seite 3 bereits erwähnt, so gewählt, dass die einzelnen Schritte nach den verschiedenen Rollen des System gegliedert sind.

Im ersten Unterabschnitt 3.1 auf Seite 34 sollen alle Anforderungen an das neue System und damit die Anforderungen an unsere Projektarbeit skizzenhaft aufgezeigt werden. Im nächsten Schritt soll die Modellierung, und damit die ersten Überlegungen zum Design der Software, der gerade beschriebenen Anforderungen ausführlich besprochen werden. Im darauffolgenden Abschnitt soll die Objektorientierte Modellierung (OOM) angesprochen werden. Es zeigt wie mit Hilfe der Unified Modelling Language (UML) das erste Design des System ensteht. Im Unterabschnitt 3.3 auf Seite 41 steht die Umsetzung des Systems und der verschiedenen Rollen im Vordergrund desweiteren zeigt es wie die Anforderungen umgesetzt sind. Der Unterabschnitt 3.4 auf Seite 46 geht näher auf die Entwicklung des DB-Systems zur Informationsverwaltung ein. Der letzte Abschnitt im Teil der Modellierung ist unter Unterabschnitt 3.5 auf Seite 54 zu finden und beschreibt die komplette Konfiguration von Server und Infrastruktur, also der kompletten Laufzeitumgebung, an der Schillerschule.

3.1 Anforderungen an das System

Außer den Problemen die mit dem alten Tool bestehen gibt es zusätzlich weitere neue Anforderungen an das System, die den Schulalltag und vor allem die Kurswahl und -verwaltung einfacher machen sollen. Es werden dem Leser zunächst die Anforderungen an das neue Tool aufgezeigt um später Rückschlüsse auf die Funktionalität und die gesamte Umsetzung ziehen zu können. An dieser Stelle wird angemerkt, dass es sich hierbei um die originalen Anforderungsspezifikationen vor Projektbeginn handelt.

Allgemeine Systemanforderungen

- webbasierte Zugang zum System über das World Wide Web (WWW) und im Intranet
- Rollenunterstützung um Berechtigungen zu unterscheidende
- Benutzerrelevante Daten sollen angezeigt werden und bearbeitet werden können

Schüler

- Wahl von Kursen
- Einsicht in Leistungsübersicht:
  - Protfoliofunktion
  - Druck- und Exportfunktion

Kurslehrer

- Einsicht der Teilnehmerlisten von Kursen

  - Druck- und Exportfunktion

- Eingabe von Leistungsnachweisen der Kurse
  - Druck- und Exportfunktion

Klassenlehrer

- Einsicht einer Klassenübersicht
  - Druck- und Exportfunktion
- Eingabe von Leistungsnachweisen
  - bearbeiten des Portfolios der einzelnen Schüler in der Klasse
  - Druck und Exportfunktion des Portfolios, der Klassenliste und alle Schüler mit ihren gewählten Kursen in der Klasse

Admin

- Erstellung/Verwaltung des Kursangebots
- Erstellung/Verwaltung der Klassenlisten
- Erstellung/Verwaltung der Leistungsnachweise
  - Druck- und Exportfunktion

Aus den ersten Überlegungen entstand folgende Mindmap welche unter Abbildung 25 auf Seite 91 zu sehen ist.

An erster Stelle befindet sich das System mit dessen Laufzeitkomponenten und Eigenschaften. Der zweite Punkt beinhaltet Schlagwörter die auf die Interaktionsmöglichkeiten

des Systems zurückgehen. Schnell wurde klar, dass es sich um ein sehr dynamisches und stetig veränderndes System handeln muss. Außerdem mussten alle Anwendungsfälle des Systems möglichst zu Projektbeginn abgedeckt und durchdacht werden um später irreversible Änderungen zu vermeiden. Auf diesen Punkt wird allerdings in Unterabschnitt 3.2 auf Seite 36 noch näher eingegangen.

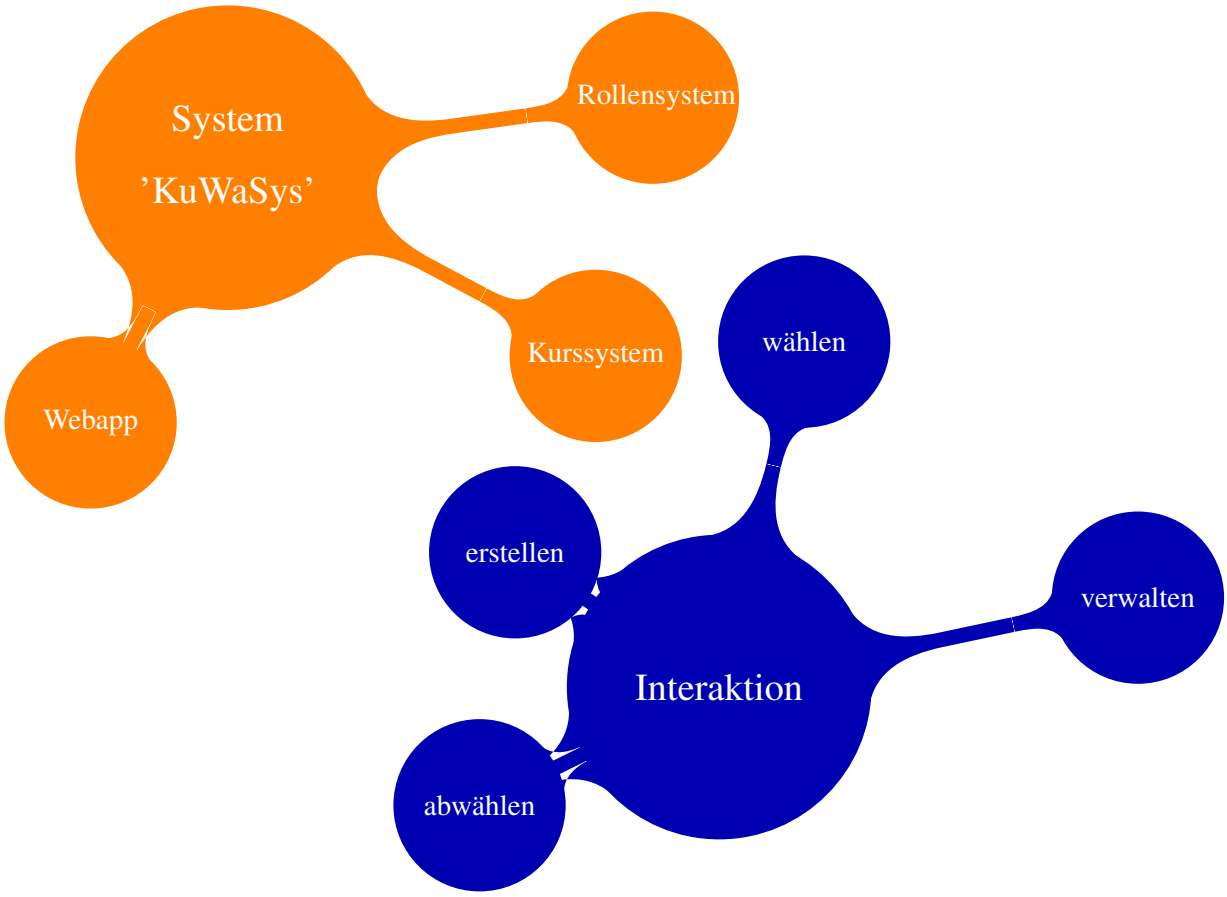


Abbildung 8: Projekt Mindmap

3.2 Objektorientierte Modellierung

Wie schon in Abschnitt 3 auf Seite 33 angesprochen, soll nun die OOM zur Sprache kommen.

Zu Beginn sollen die von uns erstellten Anwendungsfälle, die sich aufgrund genauerer Analyse der Anforderungsspezifikation des Systems ergeben haben, näher erläutert werden. Hierzu wurde die Unified Modelling Language (UML), wie schon oben beschrieben, mit Hilfe eines Use-Case-Diagramm (UC-Diagramm)s, verwendet.

3.2.1 Use-Case Modell

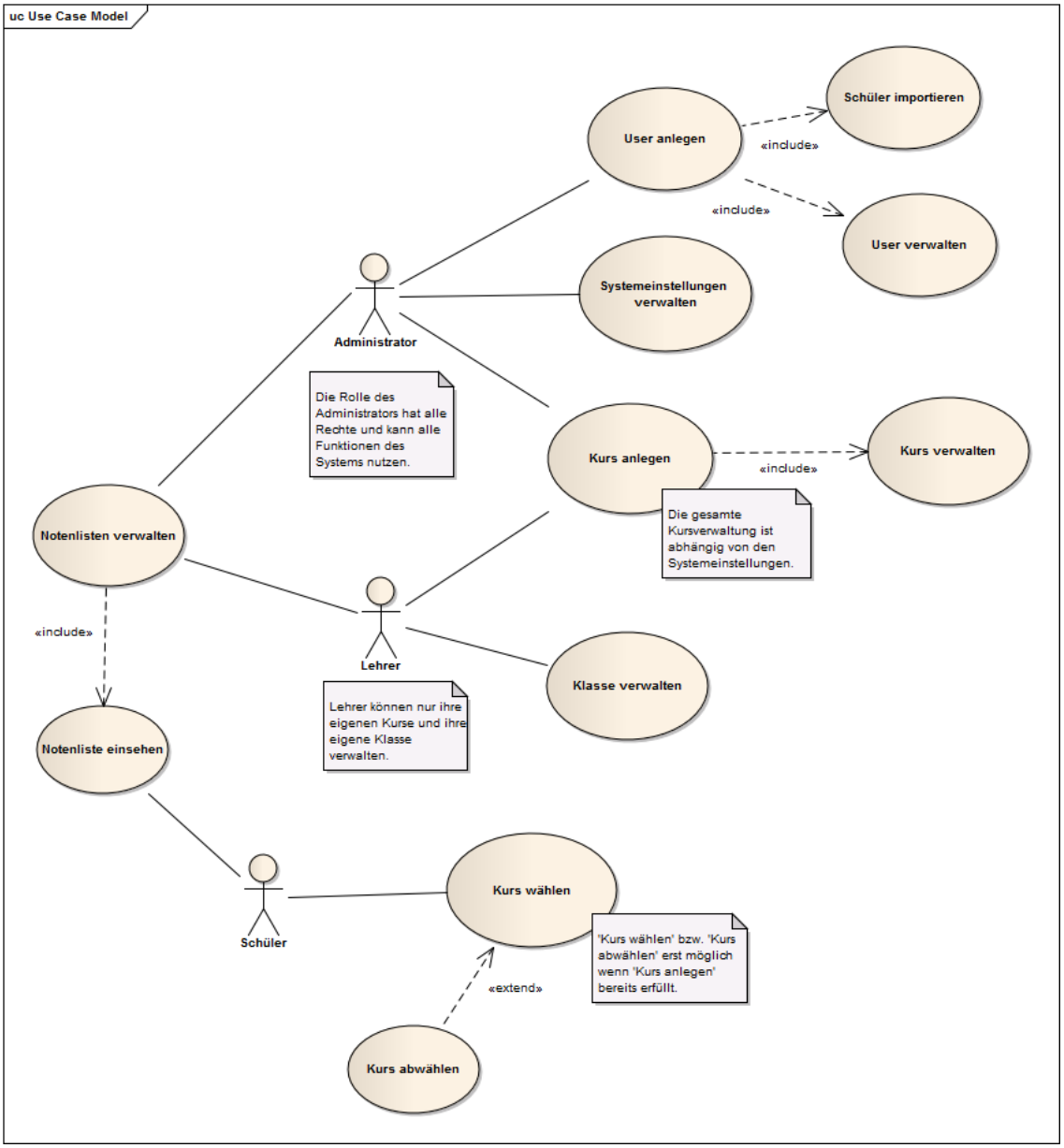


Abbildung 9: Use Case Diagramm: Rollensystem

'Ein Use-Case (UC) beschreibt die Funktionalität des Softwaresystems, die ein Akteur ausführen muss, um ein gewünschtes Ergebnis zu erhalten oder um ein Ziel zu erreichen. UCs sollen es ermöglichen, mit dem zukünftigen Benutzer über die Funk-

tionalität des Softwaresystems zu sprechen, ohne sich gleich in Details zu verlieren.' (Heide Balzert [5], [Bal10], 28).

Zusammenfassend bedeutet dies, dass ein UC-Diagramm immer dann sinnvoll ist wenn die Interaktionsmöglichkeit eines Systems, basierend auf verschiedene Aktueren, aufgezeigt werden soll. Die Akteure (dargestellt als 'Strichmännchen') im Diagramm, entsprechen genau denen, die später vom Rollensystem unterstützt werden sollen. Die Ellipsen zeigen die verschiedenen Anwendungsfälle die im System existieren. In einem so allgemeinen UC-Diagramm wird absichtlich auf kleinste Details verzichtet. So bedeutet zum Beispiel der UC 'Klasse verwalten' sowohl das Bearbeiten einer Klasse als auch das Vergeben von Noten oder andere klassenadministrative Aufwendungen.

Die gestrichelten Pfeile mit der Beschriftung '<<include>>' bezeichnen Anwendungsfälle in denen der UC, auf welchen der Pfeil zeigt, implizit vorhanden ist wenn der UC, von dem der Pfeil ausgeht, im System vorhanden ist. Das bedeutet wenn der UC 'User anlegen' also vorhanden ist, auch die UCs 'Schüler importieren' oder 'User verwalten' verwendet werden können. Ein erster UC muss also immer erfolgen während ein zweiter (oder noch mehr) optional ausgeführt werden können.

Im Gegensatz hierzu bedeutet der gestrichelte Pfeil mit der Beschriftung '<<extend>>' von dem der Pfeil ausgeht, dass dieser UC nur dann im System überhaupt vorhanden ist, wenn der UC auf welchen der Pfeil zeigt im System vorhanden ist. Der UC 'Kurs abwählen' ist also nur dann vorhanden wenn der UC 'Kurs wählen' im System ausgeführt wurde. Der erste UC wird durch den zweiten also erweitert.

[5] Informatikerin, die sich vor allem mit Fragen zum Thema Softwareengineering und -design beschäftigt. Derzeit ist sie Dozentin an der Fachhochschule Dortmund.

3.2.2 Statisches Analysemodell

Im nächsten Schritt unserer Modellierung ist das statische Analysemodell, unter Abbildung 10 auf Seite 41 zu sehen, entworfen worden. Es stellt erste Überlegungen der Softwarearchitektur, mit konkreten Objekten inklusiver ihrer Attribute, dar und bildet ebenfalls die Beziehungen von Objekten zueinander ab. Genau genommen handelt es sich hierbei um ein 'abgespecktes' Klassendiagramm, in welchem die grobe Softwarestruktur erkennbar sein soll.

Die rechteckigen Formen stellen Klassen dar, die oben als Beschriftung ihren Namen tragen, unten die Attribute die zu ihr gehören. Die einfachen Linien sind Assoziationen zwischen den Klassen und können als Beziehungen interpretiert werden. Sie tragen einen Rollennamen und eine Multiplizität, um nachvollziehen zu können um wieviele Objekte einer Klasse es sich später mindestens und maximal handelt. Die Rechtecke mit der Beschriftung `<<dataType>> + String` stellen selbstdefinierte Datentypen dar. Die Besonderheit in diesem Diagramm ist die Komposition (Assoziation mit einseitg schwarzer Raute). Sie sagt aus dass die Beziehung zwischen zwei Klassen einer starke Form der Aggregation entspricht. Die Teilklasse (Notenliste) kann also nur bestehen, solange die Aggregatklasse (Kurs) auch besteht. Würde, angewendet auf dieses Beispiel, ein Kurs gelöscht werden, würde auch der dazugehörige Notenlisteneintrag gelöscht werden. ([**?**], 18)

In unserem Projekt entstanden zum Zeitpunkt des Softwareentwurfs 3 Klassen, die später für eine Interaktion mit dem System benötigt werden und dementsprechend die wichtigsten Objekte abbilden. Die Klasse für die Notenübersicht und für Kurse. Die verschiedenen Rollen wurden als Unterklassen der Klasse 'User' modelliert. Einzelne Datentypen, so bspw. für Daten zur Zeiterfassung (Datum) und zur Definition einzelner konstanter Strings (Name), wie die Rollen, wurden zur Vereinfachung vorgesehen. Die Rollennamen sowie die Multiplizitäten der Assoziationen dürften selbsterklärend sein.

In Abschnitt Abschnitt 4 auf Seite 58 wird später näher auf die restlichen Handler- und Helper-Klassen eingegangen

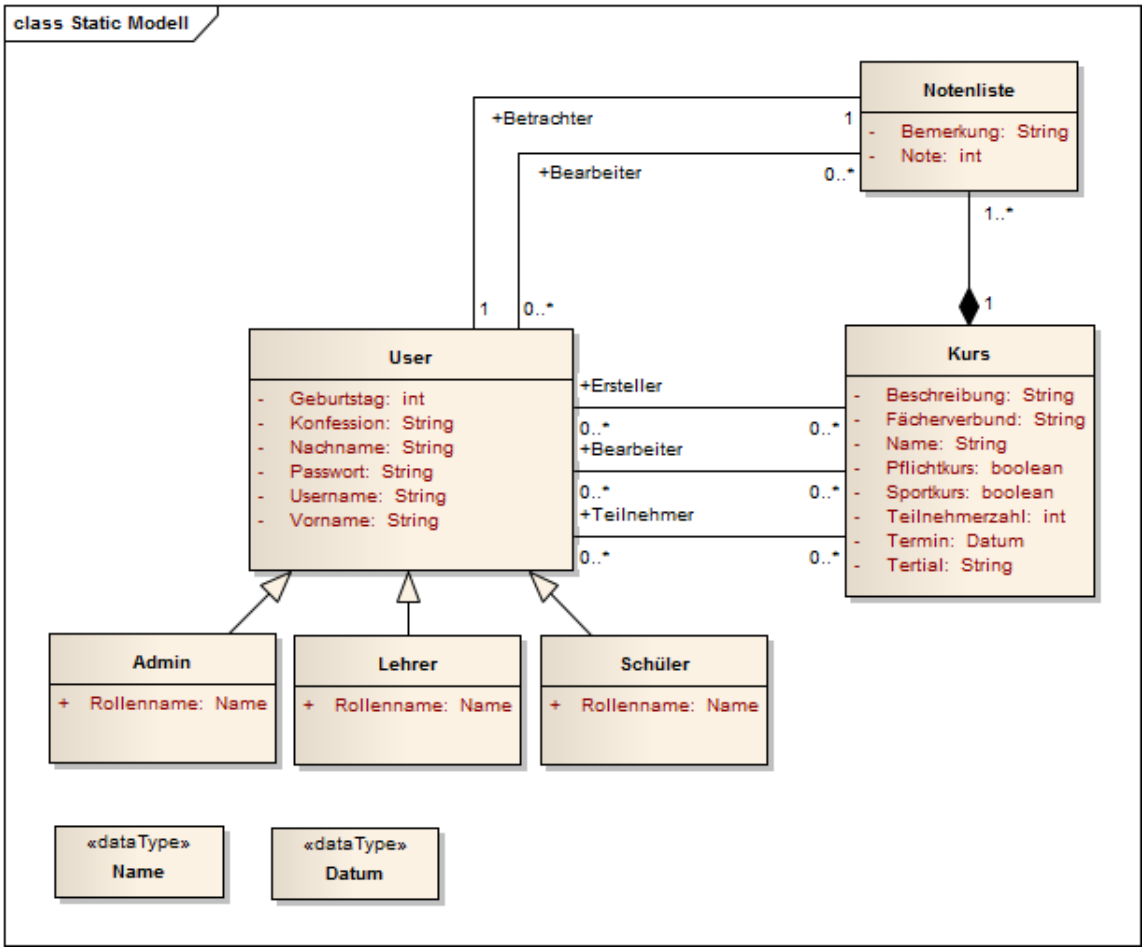


Abbildung 10: Statisches Analysemodell

3.3 Benutzerschnittstellen und Rollensystem

Das Kurswahlsystem der Schillerschule soll, laut Anforderungen, webbasiert mit Hilfe von verschiedenen Rollen konfiguriert und benutzt werden können. Das System muss hierzu simultane Interaktionen, eines jeden Rollentyps, mit dem System zulassen.

Per Aufgabendefinition liegen die vier Rollen, wie in Unterabschnitt 3.1 auf Seite 34 bereits behandelt, zugrunde:

- Schüler
- Kurslehrer
- Klassenlehrer
- Admin

Anhand der bereits vorgestellten OOM sind, vor der Implementierung des Systems, die Rollenrechte und -richtlinien welche in den folgenden drei Unterabschnitten genaue beschrieben werden, modelliert worden.

3.3.1 Die Rolle der Schüler

Bei der Rolle der Schüler handelt es sich um die einfachste des Systems. Sie hat die wenigsten Rechte und kann hauptsächlich nur passiv, Ausnahme bildet hier die Kurswahl selbst, mit dem System interagieren. Die Schüler welche das Kurswahlsystem erstmalig benutzen, befinden sich in der 7. Klasse. Jeder Schüler wird bis zum Abschluss der 9. Klasse das System benutzen. Ein Schüler soll selbstständig seine Kurswahl für das jeweilige bevorstehende Tertial eines Schuljahres treffen können, dabei müssen bestimmte Abhängigkeiten eingehalten werden:

1. 6 unterschiedliche Kurse pro Tertial
2. 18 Kurse im Schuljahr
3. 54 Kurse bis zur Vollendung des 9. Schuljahres

3.3.2 Die Rolle der Lehrer

Der Rolle des Lehrers ist hingegen schon weitaus mehr Verantwortung auferlegt. Das System unserer Webandwendung unterscheidet allerdings zwischen zwei verschiedenen Arten von Lehrern grundsätzlich nicht aufgrund einer Rollendefinition. Die Befugnis eines Lehrers sind lediglich abhängig von Werten die in der DB gespeichert werden. Ein Klassenlehrer wird nur dann ein Klassenlehrer wenn für ihn eine Abhängigkeit zu einer Klasse besteht. Erst dann kann er die Funktionen, die einenm Klassenlehrer zu Verfügung stehen, nutzen. Ein Klassenlehrer muss auf die ihm zugerordnete Klasse zugreifen und alle Kurse eines jeden Schülers einsehen können.

Das gleiche Prinzip gilt für Kurslehrer. Beim anlegen eines Kurses (eine Funktion die jedem Lehrer generell zur Verfügung steht) wird dieser Kurs dem Lehrer fest zugeordnet und wird somit zum Mitverantwortlichen der Kursverwaltung. Der Kurslehrer muss ebenfalls die Schüller einsehen können die sich in seinem Kurs befinden. Allerdings kann er nur ein Protokoll und eine Notenliste für Schüler in seinem Kurs führen.

In Abhängigkeit zur administrativen Systemverwaltung, auf die im nächsten Teil-Block eingegangen wird, hat der Lehrer nun die Rechte seine Kurse zu verwalten.

Zur Vereinfachung der Handhabung des Systems ist bereits an diesem Punkt der Modellierung an ein internes Protokollierungssystem gedacht worden. Es soll später die Kommunikation von Kurslehrern mit Klassenlehrern sowie die Kommunikation von Lehrern mit Schülern vereinfachen. Im Rahmen unserer Projektarbeit wurde allerdings eine solche Funktionalität im System nicht implementiert.

3.3.3 Die administrative Rolle des Systems

Die administrativen Rechte des kompletten Systems stehen selbstverständlich nur dem Administrator zur Verfügung. Die Hauptaufgabe dieser Rolle im System ist es neue User ins System aufzunhemen und diese gegebenfalls zu Bearbeiten. Das gilt für das Hinzufügen und Bearbeiten von Schülern und Lehrern gleichermaßen, beide Rollen haben also nicht die Möglichkeit sich selbst zu Verwalten.

Darüber hinaus verwaltet der Admin den Status des Systems, das aktuelle Schuljar und die dazughörigen Tertiale. Außerdem ist er der Hauptverantwortliche der Kursverwaltung. Bevor ein Kurs stattfinden kann muss der Admin den Kurs aktivieren und für die Kurswahl zulassen. Eventuell müssen von ihm bestimmte Attribute eines Kurs noch angepasst werden können. Ein Admin kann außerdem Fächerverbünde erstellen, bearbeiten und erstellte Kurse einem Fächerverbund zuordnen.

3.3.4 Funktionalität des Systems

Die Einteilung eines kompletten Schuljahrs geschieht jeweils in Tertiale. Dies kann grafisch in Abbildung 11 auf Seite 45 nachvollzogen werden.

Es war also nötig im administrativen Bereich des Systems eine Funktionalität vorzusehen, die es dem Admin ermöglicht das Schuljahr 'weiterzuschieben' und das entsprechende Tertial zu zu aktivieren. Diese Verwaltungstätigkeit ist unumgänglich mit der Kurswahl verknüpft und muss für jede neu angepasst werden. Entscheidungen die hier während der Modellierung getroffen wurden, wurden im Hinblick auf eine möglichst einfaches UI getroffen.

Jeder Kurs, der gewählt werden kann, gehört einem übergeordnetem Fächerverbund an, wie in Abbildung 12 auf Seite 46 zu sehen ist. Allerdings gibt es noch weitere Ei-

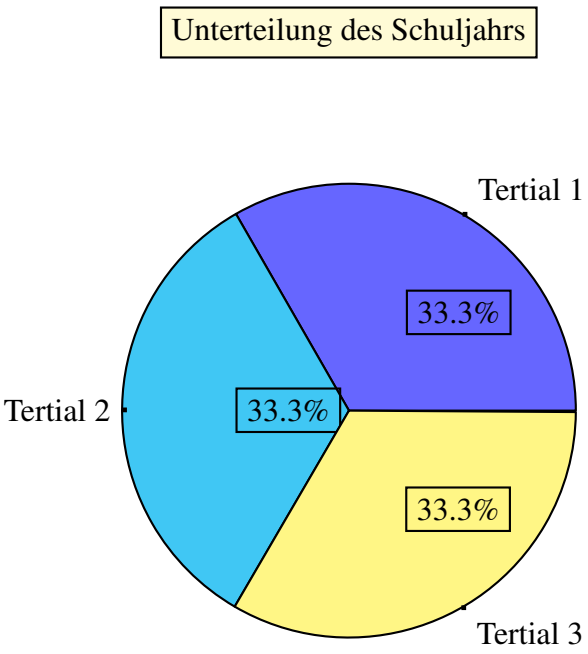


Abbildung 11: Einteilung des Schuljahrs

genschaften die Kurse erfüllen können. Ein Kurs kann ein Religionskurs sein, welcher für eine bestimmte Konfession ausgerichtet ist oder ein Sportkurs. Ferner müssen Kurse als Pflichtkurse signifiziert werden können. Aufgrund der Fülle an Informationen die verarbeitet werden müssen, war es in der Phase der Modellierung ebenfalls sehr wichtig die Datenstrukturen und deren Abhängigkeiten möglichst einfach zu halten, um in der späteren Implementierungsphase eine unkomplizierte Benutzerschnittstelle entwicklen zu können.

Außer den beiden vorgestellten Datenverarbeitungen existieren selbstverständlich noch weitere, die vor allem im Sinne der Übersichtlichkeit des System zum Tragen kommen. Dabei handelt es sich allerdings um Daten, die dynamisch während der Laufzeit erzeugt werden und deshalb ebenfalls in Unterabschnitt **??** auf Seite **??** genauer besprochen werden. Im Gegensatz zu den dynamisch generierten Daten während der Laufzeit des Systems ist für alle anderen Daten zur Informationsverarbeitung eine Speicherung in einer Datenbank unabdingbar. Im folgenden Unterabschnitt soll deshalb näher auf

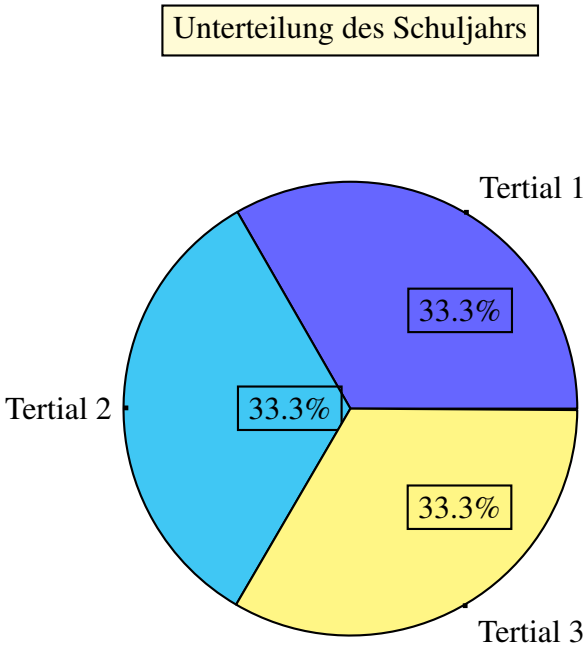


Abbildung 12: Einteilung des Schuljahrs

die Modellierung und Umsetzung des verwendeten Informationssystems eingegangen werden.

3.4 Datenbankmodellierung

Für die DB-Modellierung wurde als erstes ein Entity-Relationship Modell (ER-Modell) skizziert, welches die Abhängigkeiten und Beziehungen zueinander verdeutlichen soll. Dieses ist in Unterabschnitt 3.4.1 auf Seite 47 abgebildet. Anschließend wurde das gesamte ER-Modell in ein relationales Modell transformiert. Mit dem entstandenen Relationalen Modell, welche in Unterabschnitt 3.4.2 auf Seite 51 dokumentiert ist, konnte anschließend auf dem Server die DB inklusive der nötigen Tabellen erstellt werden.

3.4.1 Entity-Relationship Modell

Die Abbildung 13 auf Seite 48 zeigt das oben erwähnte ER-Modell nach Peter Pin-Shan Chen [6] ([Che76], 3 ff. bzw. [Vos08], 60 ff.).

Aufbau des ER-Modells:

Bei den blauen Rechtecken handelt es sich in diesem Modell um sogenannte Entities (Entity-Typen), die sind Dinge die in der DB als solche abgebildet werden sollen. Sie stellen eine eigene Tabelle dar. Die gelben Ellipsen die diese umgeben, sind die Attribute (Attribut-Typen) der Entities, sie veranschaulichen die Daten welche Entities enthalten (können). Die roten Rauten bezeichnen Beziehungen (Beziehungs-Typen) die zwischen Entities herrschen.

Das grüne Rechteck ist ein Spezialfall des 'Kurs'-Entities. Es wird als eigene Tabelle in der DB dargestellt, ist im weitesten Sinne allerdings einer Aversion des 'Konfessions'-Attribut des 'Kurs'-Entities. Denkt man in diesem Fall an die UML, so wäre an dieser Stelle wohl eine 'Enumeration' als eigener Datentype in Frage gekommen ([**?**], 10 - 11).

[6] Informatiker, der 1976 das ER-Modell entwickelte. Er gilt heute als Pioneer der OOM.

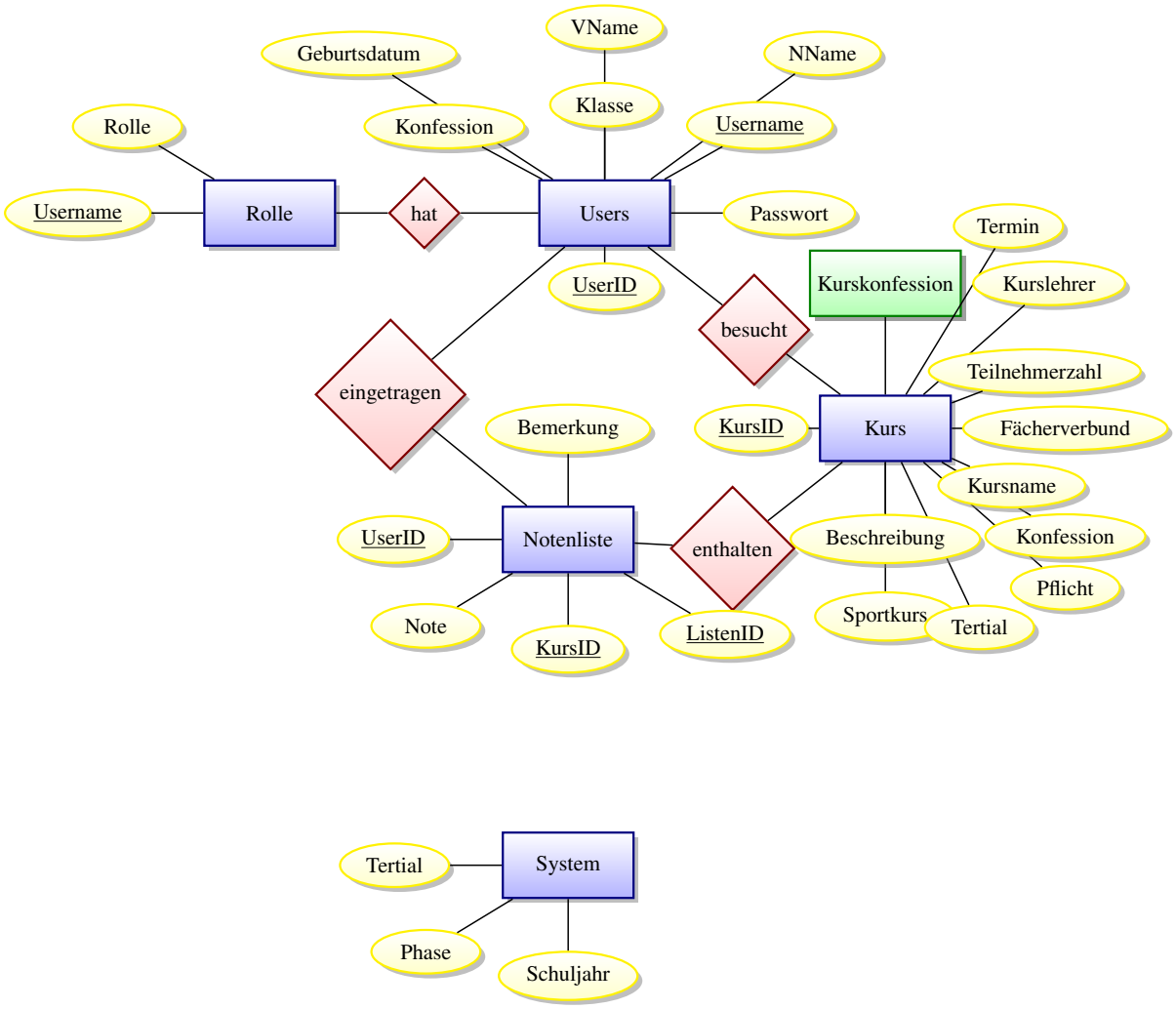


Abbildung 13: Entity-Relationship Modell

Interpretation des ER-Modells:

Um die Interpretation, also den Gedankengang der DB-Modellierung, zu verdeutlichen macht es Sinn, die Beziehungen genau auszuformulieren. Dabei werden alle Beziehungen zu jedem Entity betrachtet, begonnen mit dem Entity. Zusätzlich sollen die

Multiplizitäten mit einfließen, um die Abhängigkeiten zu verdeutlichen und um die Transformation in ein Datenbankmodell zu erleichtern.

Die Schreibweise dieser Multiplizitäten ist wie folgt definiert:

Sei E_A das erste und E_B das zweite Entity. Die Beziehung beider Entities ist definiert als R_{AB}. Die erste Multipliziät $(0, N)$ gibt Auskunft über die Häufigkeit von E_A in der geltenden Beziehung zu E_B. Die zweite $(1; M)$ über die Häufigkeit von E_B zu E_A.

Man kann also schreiben:

E_A ist $(0; N)$ in Beziehung R_{AB} zu E_B

oder

E_B ist $(1; M)$ in Beziehung R_{AB} zu E_A

Wobei die erste Zahl bei der Angabe der Multiplizität, also die vor dem Semikolon (';') für die minimalste, die zweite Zahl für die maximalste Gültigkeit unter der bestendenen Bedingung steht.

Begonnen werden soll die genauere Betrachtungsweise mit dem Entity 'User':

1. User - Rolle (beidseitig):
   - einem User ist genau $(1; 1)$ Rolle zugeteilt
   - einer Rolle hingegen können $(0; N)$ User zugeteilt sein
2. User - Kurs
   - ein Schüler wählt $(0; N)$ viele Kurse

- ein Lehrer erstellt/verwaltet $(0; M)$ viele Kurse

3. User - Notenliste:

   - ein Schüler hat $(1; 1)$ Eintrag in der Notenliste für jeweils einen fest zugeordneten Kurs (Vrgl. Kurs - Notenliste)

Konkretisieren wir nun das Entity 'Kurs':

1. Kurs - Notenliste:

   - ein Kurs besitzt genau $(1; 1)$ Eintrag pro Kurs in der Notenliste für einen Schüler (Vrgl. User - Notenliste)

2. Kurs - Kurskonfession

   - ein Kurs hat genau $(0; 1)$ Konfessionszugehörigkeit

3. Kurs - User

   - ein Kurs wird von $(0; N)$ vielen Schülern gewählt
   - ein Kurs wird immer genau $(1; 1)$ Lehrern zugeteilt

Für das letzte Entity der Dreier-Beziehung 'Notenliste' gelten lediglich die bereits beschriebenen Abhängigkeiten und Beziehungsverhältnisse. Zur Verdeutlichung sollen diese jedoch nochmals aufgeführt werden.

1. Notenliste - User:

   - eine Notenliste hat im Bezug auf genau einen Kurs $(0; N)$ Einträge für einen User

2. Notenliste - Kurs:

- Ein Schüler wählt $(1; N)$ viele Kurse, ein Kurs wird von $(1; N)$ vielen Schülern besucht/gewählt.

Das Entity 'System' besitzt keine Beziehungen innerhalb der Datenbank, weshalb auf eine ausführliche Interpretation verzichtet werden kann. Die Darstellung dieses Entities wird innerhalb der DB sowieso über eine eigene Tabelle umgesetzt.

Der nächste Schritt ist nun die Transformation von ER-Modell in das Relationale DB-Modell.

3.4.2 Relationales Modell

Bei der Beschreibung des Relationalen Modells der KuWaSys-DB ist das Hauptaugenmerk auf die komplette Datenverarbeitung gelegt, also vor allem auch Implementierungen für Vorgänge die für den Benutzer des Systems nicht unmittelbar zu sehen sind. Daten die für die Oberfläche und die einzelnen Benutzerschnittstellen eine tragende Rolle spielen sollen unter (Abschnitt Benutzerschnittstellen) gesondert behandelt werden und werden im Laufe dieses Kapitels nur kurz angesprochen.

Die Vorüberlegungen zur Transformation von mehrwertigen Attributen von Entities sind bereits abgeschlossen. Prinzipiell kann man mit der Transformation, wie folgt vorgehen ([Vos08], 104 ff.):

1. Jedes Entity wird in eine relationale Form gebracht
2. Jeder Beziehungs-Typ ebenfalls, es sei denn:
    - es handelt sich um eine zweistellige $1:1$-Beziehung
    - es handelt sich hierbei um eine $1:N$-Beziehung

Die Beschreibung $1:1$- bzw $1:N$-Beziehung bedeutet in diesem Fall allerdings nicht wie zuvor, ein Minimum auf der linken und das Maximum auf der rechten Seite. Hierbei werden nur noch die maximalen Werte der beiden Multiplizitätsangaben berücksichtigt. Dies gilt analog für alle Angaben der Multiplizitäten.

Sollte bei der Transformation $Punkt\ 2)$ eine Rolle spielen, müssen Attribute in bereits bestehende Relationsschemata aufgenommen werden. Folgende Regeln treten dann in Kraft:

1. Zweistellige $1:1$-Beziehung
    - ein Entity stellt selbst ein Relationsschema dar
    - Attribute des zweiten Entities werden ebenfalls in dieselbe Tabelle gespeichert
2. Zweistellige $1:N$-Beziehung
    - ein Entity stellt ein eigenes Relationsschema dar
    - erste Möglichkeit: die Attribute des Entities welches die maximale Multiplizität von 1 aufweist wird hingegen dem Realtionsschema mit der maximalen Multiplizität von N in Form von Fremdschlüsseln zugeschrieben
    - zweite Möglichkeit: es wird ein eigenes Relationsschema für die Beziehung der beiden Entities angelegt. Dieses neu entstande Schema erhält dann Attribute, welche widerum Fremdschlüssel der beiden anderen Entities sind

Die Transformation vom ER-Modell ins Relationale Modell (zur bildhaften Darstellung ist die Kopfzeile der dazugehörigen Tabelle auch gezeigt) sieht im Falle des Kurswahlsystems folgendermaßen aus:

Users = {(id:serial, nachname:character, vorname:character, geburtstag:character, konfession:character, klasse:character, username:character, passwort:character)}

ID	**NName**	**VName**	**Geb**	**Konf**	**Klasse**	**Username**	**Passwort**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tabelle 3: Kopfzeile der User-Tabelle

Kurs = {(id:serial, name:character, kurslehrer:integer, faecherverbund:character, termin:integer, beschreibung:character, schuljahr:integer, tertial:integer, teilnehmerzahl:integer, pflichtkurs:boolean, sport:boolean)}

ID	**Name**	**Kurslehrer**	**Faecherverbund**	**Termin**	...
⋮	⋮	⋮	⋮	⋮	...

Tabelle 4: Kopfzeile der Kurs-Tabelle (unvollständig)

Kurs-Konfessionen = {(religionid:integer, konfession:character)}

ReligionID	**Konfession**
⋮	⋮

Tabelle 5: Kopfzeile der Kurs-Konfessions-Tabelle

Notenliste = {(id:serial, note:integer, bemerkung:character, userid:integer, kursid:integer, jahr:integer, tertial:integer)}

ID	**Note**	**Bemerkung**	**UserID**	**KursID**	**Jahr**	**Tertial**
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tabelle 6: Kopfzeile der Notenlisten-Tabelle

Rolle = {(username:character, rolle:character)}

Username	**Rolle**
⋮	⋮

Tabelle 7: Kopfzeile der Rollen-Tabelle

System = {(phase:integer, jahr:integer, tertial:integer)}

Phase	**Jahr**	**Tertial**
⋮	⋮	⋮

Tabelle 8: Kopfzeile der System-Tabelle

3.5 Konfiguration der Laufzeitumgebung und des Servers

Dieses Kapitel ist als Zwischenschritt, von der Modellierung zur Implementierung, unseres Softwareprojekts zu verstehen. Zum Einen galt es eine komplette bestehende Infrastruktur zu überblicken und zu verstehen (dieser Schritt kann als eine Art der Modellierung verstanden werden) zum Anderen musste ein komplettes neues System einwandfrei funktionsfähig eingebettet werden (zu vergleichen mit dem Schritt der Implementierung).

Sichtung der bestehenden Infrastruktur:

Die Schillerschule teilt sich mit der benachbarten Realschule am Galgenberg eine Serverinfrastruktur nach der Novell Musterlösung paedML 3.33, weitere Informationen sind unter [bw.13] zu finden. Diese beinhaltet eine virtuelle Infrastruktur VMWare ESXi auf der ein SuSE Linux Enterprise Server (SLES) Novell Server gehostet ist.

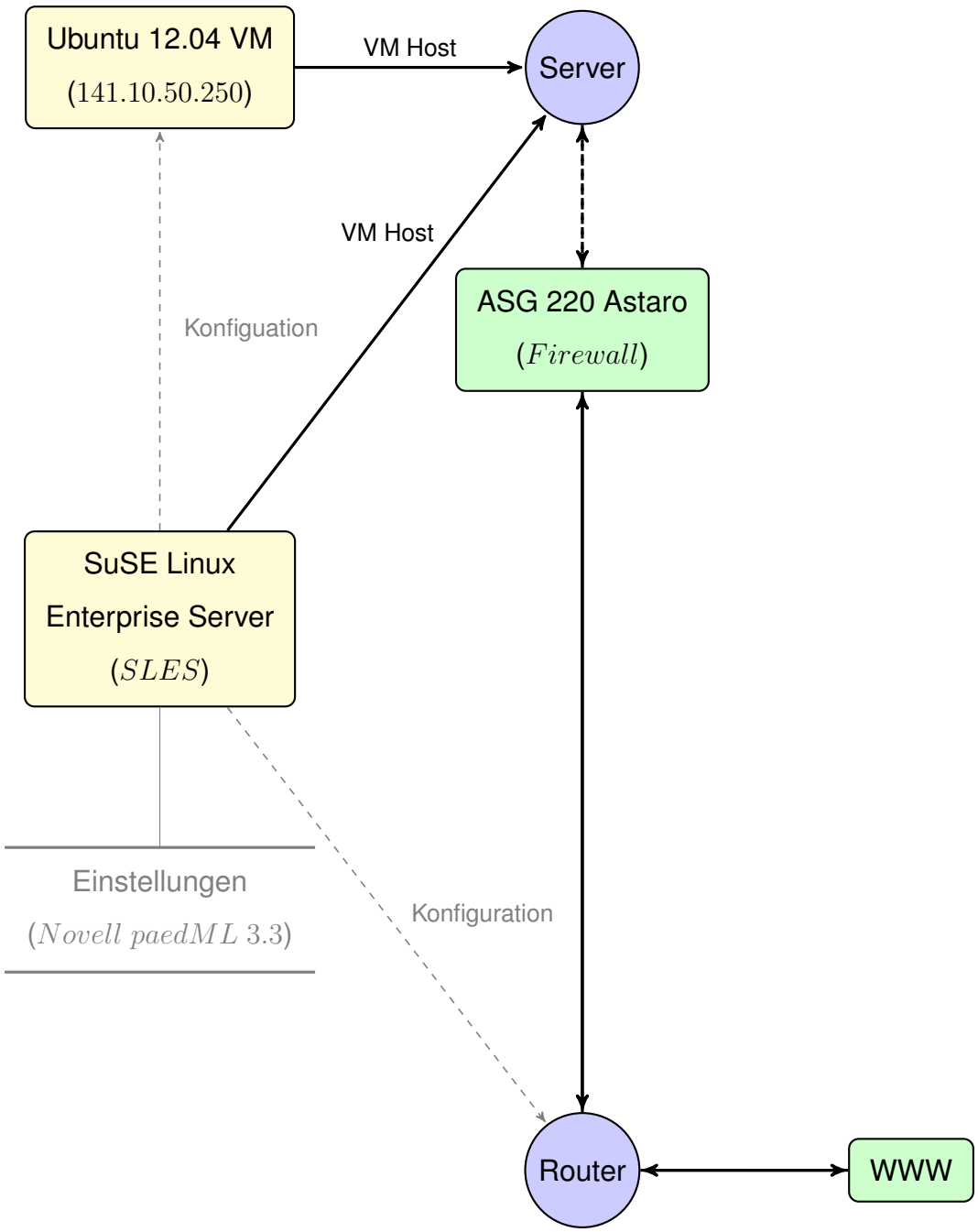


Abbildung 14: Netzwerkstruktur der Schillerschule Aalen mit DMZ

Prinzipiell wäre eine Verwendung dieser virtuellen Appliance zum Hosting der Webapplikation (Webapp) möglich. Um jedoch die Systemsicherheit zu erhöhen ist eine

weitere virtuelle Appliance, die nur das Kurswahlsystem bereitstellt die bessere Wahl. Ubuntu Server 12.04 LTS [7] läuft auf diesem virtuellen Rechner, der sich wie der oben genannte SLES in der DMZ der virtuellen Netzwerkinfrastruktur befindet.

Integration in die Infrastruktur:

Ein PostreSQL-Server dient zur Datenhaltung und ein Apache Tomcat 7 zur Auslieferung der Webapp. Nach Konfiguation (und überfälligem reboot) der Astaro Firewall im Keller der Schule ist die Webapp jetzt über das Intranet an allen Rechnern im Schulnetz erreichbar (`http://192.168.1.222:8080/kuwasys20`). Die Erreichbarkeit über das Internet ist seit der Freischaltung der entsprechenden Ports auf den BelWü-Server und umgekehrt gegeben (`http://141.10.50.250:8080/kuwasys20`). Diese Adressen werden auf der Homepage der Schillerschule und im Intranet verlinkt, sodass keine weitere Maskierung wie Subdomains oder lokale DNS-Einträge notwendig ist. Die Admnistration des Ubuntu Servers kann im Intranet von einer VMWare Management Console aus erfolgen, für schnelles Eingreifen wurde ein SSH Zugang eingerichtet, der im Internet erreichbar ist.

In Abbildung 14 auf Seite 56 ist die neue Struktur des Netzwerks dargestellt.

Die Einrichtung eines Backups war für unser Projekt nicht notwendig, da die Schillerschule selbst schon über ein funktionsfähiges Backupsystem verfügt. Hierbei wird in bestimmten Intervallen die komplette Festplatte des Servers gespiegelt, und somit auch die virtuellen Maschinen. Somit ist die Garantie gegeben, dass auch das Kurswahlsystem einem ständigen Sicherungsvorgang unterliegt und im Notfall wiederhergestellt werden kann.

[7] http://www.ubuntu.com/

4 Implementierung

In einem System, in welchem ein Multi-User Betrieb möglich sein soll, ist das Design der Oberfläche und das der einzelnen Benutzerschnittstellen unweigerlich eng miteinander verknüpft. Es müssen in der Phase der Implementierung bereits exakte Schnittstellen definiert und strukturierte Oberflächen skizziert worden sein um spätere Korrekturen gering zu halten oder um sie zu vermeiden.

In den folgenden zwei Abschnitten sollen grundlegende Implementierungsgedanken besprochen werden, die mit den Anforderungen an das System in erster Linie nichts zu tun haben. Zuerst soll die Art und Weise der Umsetzung der Benutzerschnittstellen und des Designs näher erklärt werden. Danach sollen elementare Datenstrukturen die zum Einsatz kamen und in JSF implementiert wurde näher erläutert werden.

Nach den beiden einführenden Abschnitten wird dem Leser detailiert dargestellt wie die zu bewältigenden Systemanforderungen in JSF umgesetzt wurden. Dabei werden Quellcodeausschnitte sowie Diagramme zum Einsatz kommen die dem Leser das Verstehen erleichtern sollen. Da während allen Phasen der Umsetzung des Projekts auch immer die Modularität des gesamten Systems im Vordergrund stand soll hier nicht kleinlichst genau erklärt werden was im Quellcode steht, sondern darauf eingegangen werden, wie das zu lösende Problem angegangen wurde und schlussendlich welche wichtigen Bausteine zu tragen kamen. An dieser Stelle soll außerdem nochmals die Wichtigkeit der oben besprochenen Datenbankmodellierung erwähnt werden. Gründe für die verschiedenen Umsetzungen der Modellierung sollen im Abschnitt der Implementierung dieser Ausarbeitung nicht mehr näher besprochen werden. Es wurde jedoch viel Wert darauf gelegt die Schritte der Implementierung gut und verständlich zu formulieren und darzustellen.

4.1 Benutzerschnittstellen und Oberflächendesign

Wir haben uns für ein simples und einfach zu verstehendes Oberflächendesign entschieden, welches allerdings den Design Aspekten der Corporate Identity (CI) erfüllen sollte. Im allgemeinen wird beim Screendesign bestimmten Regeln gefolgt, welche durch das gewählte Gestaltungsraster festgelegt werden.

Hierzu wurden von uns folgende Bereiche festgelegt:

- Kopf- oder Bannerbereich mit Logo
- Navigationsbereich bzw. Unternavigation
- Arbeitsbereich
- Impressum/Hinweise

Der Hauptaufbau dieser Seiten, auf welche im folgenden näher eingegangen wird, wurden mit Templates, wie es bereits in Unterabschnitt 2.4.2 auf Seite 19 angesprochen wurde, umgesetzt. Der allgemeine Aufbau soll im folgenden besprochen werden, der nachstehende Quellcodeausschnitt zeigt einen Teil des Templates das von uns zur Gestaltung der Seiten verwendet wurde:

Code 16: Template-Facelet, dass grundelegenden Aufbau der Seite beschreibt

```
1 <h:head>
2         <title>Kurswahlsystem der Schillerschule Aalen - <ui:
             insert name="title" /></title>
3         <ui:insert name="headers" />
4 </h:head>
5 <h:body>
6 <h:outputStylesheet library="css" name="style.css"/>
7         <t:div id="container">
```

```
8                 <h:graphicImage library="img" name="header.jpg
                     " alt="HeaderImage"/>
9                 <t:div id="titlebar">
10                        <h:panelGrid columns="3" style="text-
                             align:right;␣width:980px;">
11                             <ui:insert name="title" />
12                             <h:outputText style="color:
                                  lightgray;" rendered="#{
                                  kuwasys.userRole()==null?
                                  false:true}" value="
                                  Schuljahr␣#{kuwasys.year
                                  }/#{kuwasys.year+1}␣|␣
                                  Tertial:␣#{kuwasys.tertial}
                                  ␣|␣#{kuwasys.systemPhase()}
                                  "/>
13                             <h:outputText style="color:
                                  lightgray;" rendered="#{
                                  kuwasys.userRole()==null?
                                  false:true}" value="#{
                                  userBean.showUsername()}␣|␣
                                  #{request.remoteUser}␣
                                  &#8834;␣#{kuwasys.userRole
                                  ()}"/>
14                        </h:panelGrid>
15                </t:div>
16                <h:panelGrid columns="2">
17                <ui:include src="/navigation.xhtml"/>
18                <t:div id="content"><ui:insert name="content"
                     /></t:div>
19                </h:panelGrid>
20        </t:div>
```

```
21          <t:div id="footer">Kurswahlsystem - Schillerschule
               Aalen - MI-Projektarbeit im WS12/13 HTW-Aalen -
               Christian Silfang/Michael Sch tz </t:div>
22  </h:body>
```

Auffallend ist vor allem der Kopf der Seite, welcher den Wiedererkennungswert (Vrgl. hierzu die Webiste der Schillerschule [8]) ganu im Sinne des Corporate Identity (CI)s steigern soll. Hierzu wurde die Grafik lediglich transparenter gehalten als ihr Original und hat ganz im Stil der Schule die Überschrift erhalten wie in Abbildung 15 auf Seite 61 zu sehen ist.

Abbildung 15: KuWaSys: Banner des Systems (Header)

Die Navigation und deren Unternavigationspunkte sind im linken Bereich der Webiste angeordnet und unterstützen den User visuell mit Hervorhebungen, wie in Abbildung 17 auf Seite 63 dargestellt ist, bei der Arbeit mit dem System. Nötige Unternavigationspunkte, falls diese vorhanden sind, öffnen sich automatisch beim Klick auf ein übergeordnetes Menüelement, sodass die komplette Menüstruktur handlich und kompakt dargestellt erscheint und mit einem Blick erfasst werden kann.

Der Arbeitsbereich wird in der Mitte der Seite dargestellt, unterhalb des Kopfbereichs und rechts der Navigation. Dieser wird abgetrennt durch blaue Balken (links zur Navigation sowie oben zum Kopfbereich) um die Einteilung der Seite für die Benutzer des Systems eindeutig und übersichtlich zu halten. Jede Interaktion mit dem System wird in diesem Bereich der Website dargestellt.

[8] http://www.schillerschule-aalen.de

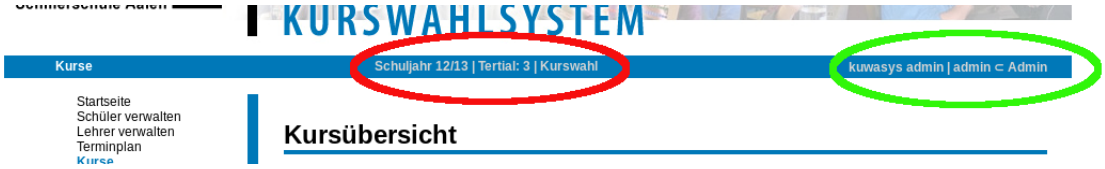


Abbildung 16: KuWaSys: Informationsbalken (oberer Bereich)

Zur erwähnen ist zudem noch der Trennbalken nach oben welcher zusätzlich als Informationsanzeige für den User verwendet wird und der Fußbereich welcher Informationen zur Website enthält. Die obere Anzeige, hier werden Informationen zum User selbst (Voller Name, Username und Rolle im System) und Informationen zum aktuellen Status des Systems (aktuelles Schuljahr und aktuelles Tertial) angezeigt. Die Abbildung 16 auf Seite 62 zeigt die genaue Einteilung des Infobalkens oben, rot die Informationen des Systems, grün die Informationen des Users. Der Fußbereich dient ausschließlich zur weiteren Information beim Besuch der Webiste welcher in Abbildung 18 auf Seite 63 zu sehen ist.

Die Hauptfarben des Systems wurden absichtlich blau gewählt um eine gewisse Professionalität sowie Seriösität gegenüber den Benutzern auszustrahlen. Diese ziehen sich kontinuierlich durch das gesamte System

Abbildung 17: KuWaSys: Navigations und Menüelemente (linker Bereich)

Auf ein Impressum oder auf rechtliche Hinweise konnte bei dieser Art von Webiste verzichtet werden. Erstens handelt es sich hierbei um keine kommerzielle Website, laut §5 Telemediengesetz (TMG) wird ein Impressum von 'geschäftsmäßigen Online-Diensten' benötigt, wobei hier die Vorschrift des §55 Rundfunkstaatsvertrages (RstV) besagt, dass eine Impressumspflicht nur dann besteht wenn der Inhalt der Website regelmäßig journalistisch-redaktionelle Inhalte online zur Verfügung stellt. (Gesetzesauszüge sind im Anhang unter Unterabschnitt A.4 auf Seite 2 zu finden)

Abbildung 18: KuWaSys: Informationsbalken (unterer Bereich)

Zur Strukturierung von Seiteninhalten wurden herkömmliche JSF-Standardkomponenten

verwendet. Zu den hauptsächlich verwendeten, zählen:

- `<h:outputText>`
  Element welches HTML-Text auf der Obefläche ausgibt
- `<h:outputLabel>`
  auch normaler Text, allerdings als Beschriftung von Texteingabefeldern
- `<h:inputText>`
  Element zur Texteingabe, Synonyme für ein HTML `<input>`-Tag mit type=”text”
- `<h:inputSecret>`
  Element zur verschlüsselten Texteingabe, entspricht `<input>`-Tag mit type=”password”
- `<h:commandButton>`
  Button in HTML, der Klick löst eine definierte Aktion über eine ManagedBean-Methode aus
- `<h:panelGrid>`
  Darstellung einer Tabelle, Synonym für das HTML `<table>`-Tag. Die Anzahl der Spalten wird über das `columns`-Attribut festgelegt
- `<h:panelGroup>`
  Container-Element (mehrere JSF-Tags werden zu einem zusammenfügt)
- `<h:message>`
  gibt eine Fehlermeldung für die definierte Komponenten aus (`ErrorStyle`-Attribut über Cascading Style Sheets (CSS) steuerbar)
- `<h:form>`
  stellt ein Formular dar, welches einen POST-Request per HTTP absetzt
- `<f:facet>`
  Definiert eine Facette (bspw. die Überschrift für eine Tabelle)

Außer den eben erwähnten Komponenten gibt es eine Vielzahl anderer bis hin zu selbstdefinierten welche im entwickelten System zum Einsatz kommen. Diese werden allerdings nicht näher betrachtet, da sie für das umgesetzte System irrelevant sind. Selbstverständlich kann zur grafischen Darstellung auch normale Hypertext Markup Language (HTML)-Syntax verwendet werden. Um die Strukturierung des Seitenaufbaus übersichtlich zu halten wird ebenfalls das wohl bereits bekannte Grundlagenwerkezug der Webentwicklung, die Cascading Style Sheets (CSS), für die Eigenschaften der Darstellungselemente, eingesetzt. Vorteile dieser Vorgehensweise der Datenverarbeitung resultiert in einer einheitlichen und damit gut strukturierten Darstellung der betroffenen Websites.

4.2 Informationsdarstellung und -verarbeitung

Zunächst soll die Erstellung der Datenbank, für die drei Entities User, Kurs und Notenliste, aufgezeigt werden. Da diese die Grundlage zum Speicher von Daten im System bildet. In diesem Abschnitt sollen zudem die benutzten Komponenten beschrieben werden die zur Darstellung der Informationen aus der Datenbank verwendet wurden.

4.2.1 Erstellung der Datenbank

USER-Tabelle CREATE

KURS-Tabelle CREATE

NOTENLISTE-Tabelle CREATE

4.2.2 Listen und Tabellen

Eine elementare Datenstrukturen im System sind Listen, welche dem User in Form einer Tabelle im Facelet dargestellt werden. Die Vorteile dieser Datenstruktur sind die Einfachheit der Datenhaltung sowie der Zugriff auf die Daten und Bereitstellung dieser.

Grundlegend sind drei zu unterscheidende Listen im System implementiert:

1. User-Listen (Schüler und Lehrer)
2. Kurs-Listen
3. Noten-Listen

Dabei sind die Listen lediglich nach Art des Inhalts zu differenzieren. Die für die Benutzeroberfläche relevanten Daten sollen kurz erläutert werden.

Schüler

- ID: Integer, welcher für das Wählen von Kursen und das Vergeben von Noten wichtig ist, da diese einen Fremdschlüssel in der jeweiligen Tabelle darstellt
- Vorname und Nachname: String, der den realen Namen des Users wiedergibt
- Klasse: String, zur Identifizierung welchem Klassenlehrer ein Schüler zugeordnet ist, bzw wie weit er in seiner Schullaufbahn vorangeschritten ist
- Konfession: String, zur Identifizierung des Religionsunterrichts der angeboten werden kann bzw. muss während dem Erstellen eines Kurses. (Näheres in Abschnitt Kurswahl)

Lehrer

- ID (mit der gleichen Bedeutung wie bei Schülern)
- Vorname und Nachname (ebenfalls gleich wie bei Schülern)
- Klasse: String, der festlegt, von welcher Klasse ein Lehrer Klassenlehrer ist

Kurse

- Kursnummer
- Kursname
- Kursbeschreibung
- Teilnehmeranzahl

Notenlisten

- Schülername
- Kursname
- Note
- Bemerkung

Die Implementierung der einzelnen Listen in Java, wurde über Listen vom Typ `ArrayList<List>` der Klasse java.util.ArrayList¡E¿ umgesetzt. Als nächstes soll am Beispiel einer User-Liste gezeigt werden, wie diese im System implementiert wurde. Hierzu sind zunächst drei Dinge nötig:

1. Facelet zur Darstellung
2. Managed Bean, zum Handling der Daten im Facelet
3. Klasse, die ein Objekt der jeweiligen Liste, mit den nötigen Attribute, repräsentiert

Die Darstellung der User-Übersicht wird mit Hilfe einer Tabelle umgesetzt.

Code 17: Facelet einer kompletten User-Übersicht

```
<t:dataTable id="data"
        var="users"
        sortable="true"
        value="#{userBean.users}"
        preserveDataModel="true">
    <t:column>
        <f:facet name="header">
            <h:outputText value="ID" />
        </f:facet>
        <h:outputText value="#{users._id}" />
    </t:column>
    <t:column>
        <f:facet name="header">
            <h:outputText value="Vorname" />
        </f:facet>
        <h:outputText value="#{users._vorname}" />
    </t:column>
    <t:column>
        <f:facet name="header">
            <h:outputText value="Nachname" />
        </f:facet>
        <h:outputText value="#{users._nachname}" />
    </t:column>
```

```
24 ...
25     <t:column>
26         <f:facet name="header">
27             <h:outputText value="Username" />
28         </f:facet>
29         <h:outputText value="#{users._username}" />
30     </t:column>
31     <t:column>
32         <f:facet name="header">
33             <h:outputText value="Passwort" />
34         </f:facet>
35         <h:outputText value="#{users._passwort}" />
36     </t:column>
37 </t:dataTable>
```

Das Attribut `value` der `<t:dataTable>` referenziert den Inhalt der Liste `users` in der Backing-Bean `userBean`. Diese ist jedoch in `userBean` nur als getter-Methode `getUsers()` vorhanden und kein Attribut der Klasse.

Code 18: userBean

```
1  package de.schillerschule.kuwasys20.User;
2  ...
3  /**
4   * Klasse f r User-Handling im System
5   *
6   * @author cy
7   *
8   */
9  @ManagedBean(name = "userBean")
10 @RequestScoped
11 public class UserBean implements Serializable {
```

```
12
13          FacesContext context = FacesContext.getCurrentInstance
                ();
14          private static Logger logger = Logger.getLogger(
                UserBean.class.getCanonicalName());
15          private static final long serialVersionUID = 2L;
16          DatabaseHandler dbh = new DatabaseHandler();
17  ...
18          public List<User> getUsers() {
19                  if (context.getExternalContext().isUserInRole(
                        "admin"))
20                          return dbh.listUsers();
21                  else
22                          return null;
23          }
24
25          public List<User> getSchedule() {
26                  if (context.getExternalContext().isUserInRole(
                        "admin"))
27                          return dbh.listClassesSchedule();
28                  if (context.getExternalContext().isUserInRole(
                        "lehrer"))
29                          return dbh.listClassesTeacherSchedule(
                                dbh.getUserId());
30                  else
31                          return null;
32          }
33
34          public List<User> getTeacherClass() {
35                  if (context.getExternalContext().isUserInRole(
                        "admin"))
```

```
36                          return dbh.listClasses();
37                  if (context.getExternalContext().isUserInRole(
                        "lehrer"))
38                          return dbh.listClassesTeacher(dbh.
                                getUserId());
39                  else
40                          return null;
41          }
42  ...
43          /**
44           * Username des aktuellen Users zurückgeben
45           *
46           * @return username
47           */
48          public String showUserFullName() {
49                  String username = dbh.showUserFullName();
50                  return username;
51          }
52
53          public String showUserFirstname() {
54                  String username = dbh.showUserFullName();
55                  return username;
56          }
57
58  ...
```

Die Daten werden beim Aufruf von `userBean.getUsers()` direkt aus der Datenbank geholt. Die Methode `listUsers()` ist in der Klasse `DatabaseHandler` implementiert und liefert nach abfrage der Datenbank ein `ArrayList<User>` Objekt zurück, dass von der oben gezeigten Tabellendeklaration verarbeitet wird.

Die Klasse User, die ein User-Objekt in der Liste darstellt, ist über eine `Nested Class` realisiert worden. Das bedeutet, dass die User-Klasse keine unabhängige Klasse, sondern in einer anderen Klasse integriert ist.

Im Beispiel hier ist die Klasse `User` in der Klasse `userBean` integriert.

Code 19: User Klasse als eingebettete Klasse

```
1  public class UserBean implements Serializable {
2  ...
3          public static class User implements Serializable {
4                  DatabaseHandler dbh = new DatabaseHandler();
5                  private static final long serialVersionUID = 1
                       L;
6
7                  private int _id;
8                  private String _nachname;
9                  private String _vorname;
10                 private String _geburtstag;
11                 private String _konfession;
12                 private String _klasse;
13                 private String _username;
14                 private String _passwort;
15                 private String _rolle; // default
16                 private boolean _canEdit;
17
18                 private int _grade_id;
19                 private Double _grade_note;
20                 private Double _grade_fachwissen;
21                 private Double _grade_sozial;
22                 private Double _grade_personal;
23                 private Double _grade_methodisch;
```

```
24                  private String _grade_bemerkung;
25
26                  private String _termin1;
27                  private String _termin2;
28                  private String _termin3;
29                  private String _termin4;
30                  private String _termin5;
31                  private String _termin6;
32                  private String _termin7;
33                  private String _termin8;
34                  private String _termin9;
35                  private String _termin10;
36                  private int _grade_kursid;
37                  private int _grade_jahr;
38                  private int _grade_tertial;
39
40                  public User(int id, String vorname, String
                        nachname, String geburtstag,
41                                  String konfession, String
                                        klasse, String username,
42                                  String passwort, String rolle,
                                        boolean canEdit) {
43                          _id = id;
44                          _nachname = nachname;
45                          _vorname = vorname;
46                          _geburtstag = geburtstag;
47                          _konfession = konfession;
48                          _klasse = klasse;
49                          _username = username;
50                          _passwort = passwort;
51                          _rolle = rolle;
```

```
52                          _canEdit = false;
53                  }
54 ...
55                  public int get_id() {
56                          return _id;
57                  }
58                  public void set_id(int _id) {
59                          this._id = _id;
60                  }
61                  public String get_nachname() {
62                          return _nachname;
63                  }
64                  public void set_nachname(String _nachname) {
65                          this._nachname = _nachname;
66                  }
67                  public String get_vorname() {
68                          return _vorname;
69                  }
70                  public void set_vorname(String _vorname) {
71                          this._vorname = _vorname;
72                  }
73 ...
```

4.2.3 Konflikte von Terminen

4.2.4 Bearbeitung von Daten im System

4.3 Import und Export von Datensätzen

Laut Aufgabenstellung war vom System eine einfache und schnelle Möglichkeit verlangt, Datensätze importieren bzw. exportieren zu können. Für den Export entschied man sich für CSV- und PDF-Dateien, für den Import sollten nur CSV-Dateien unterstützt werden. Im folgenden sollen dem Leser beide Mechanismen nähergebracht werden.

4.3.1 CSV-Dateien Import

Für den Import einer CSV Datei...

InputStreamReader

Tokenizer

JSF Artefakt

4.3.2 CSV-Dateien Export

OutputStreamWriter

JSF Artefakt

4.3.3 PDF-Dateien Export

OutputStreamWriter

iText Klasse (Artefakt)

JSF Artefakt

4.4 Benutzerauthentifizierung

Einer der elementarsten Vorgänge im System ist der Login-Vorgang. Es handelt sich hierbei um einen Vorgang den jeder Benutzer egal mit welcher Rolle vor der Interkation mit dem System durchlaufen muss. Zur Verifizierung eines Users am System wird sein Kürzel, welches beim Anlegen automatisch aus Vor- und Nachnamen generiert wird, sowie ein Passwort, welches ebenfalls automatisch und randomisiert generiert wird, benötigt. Die Anmeldemaske, welche zugleich auch die Startseite des Kurswahlsystems bildet, kann in Abbildung 19 auf Seite 77 angesehen werden. Die Registrierung eines Users am System kann ausschließlich durch den Admin erfolgen.

Für den Vorgang des Logins am System besitzt der Servlet-Container Tomcat 7 eine sehr hilfreiche Funktion, die Konfiguation des Database Connection Pool (DBCP)s des Apache Commons Projekts ([tom13]).

Wie bereits aus dem Unterabschnitt 3.4 auf Seite 46 bekannt ist, haben wir für unser Datenbank eine zusätzliche Tabelle 'Rolle', die mit der Tabelle 'User' in Relation steht, modelliert. Diese wird nun für die Konfiguration des DBCPs benötigt.

Der DBCP gehört zu einer Art

Anmeldung

Hallo zum Kurswahlsystem der Schillerschule Aalen!

Kürzel:

Passwort:

Anmelden

Aufgrund der Tatsache, dass die Rollen-Tabelle mit der User-Tabelle in einer Beziehung zueinander steht, kann bei der Authentifizierung also genau auf den gewollten User zugegriffen werden. Diese Tatsache machen wir uns auch im weiteren Verlauf der Systemimplementierung zu Nutze, bspw. bei der Generierung von Benutzerdaten (Unterabschnitt 4.5 auf Seite 77) oder aber um Daten zu manipulieren die mit dem User in Verbindung stehen.

4.5 Benutzerverwaltung

4.5.1 Anlegen von Benutzerdaten

Das Anlegen eines Benutzers erfordert immer die Informationen über:

- Vor- und Nachname
- Geburtsdatum
- Klasse
- Konfession

Die Rolle, die ein Benutzer im System erhält, wird durch zwei unterschiedlich Eingabeaufforderungsdesigns umgesetzt. Eines für Lehrer und eines für Schüler. Benutzer-

namen und Passwörter werden vom System anhand der eingegebenen Informationen automatisch generiert.

Abbildung 20: KuWaSys: User anlegen

Abbildung 21: KuWaSys: Lehrer anlegen

textbfSonderfunktion: CSV-Import

Eine Besonderheit der Benutzerverwaltung ist der Import von Userdaten über Comma Separated Values (CSV)-Dateien. Diese Funktion war im Sinner der Projektarbeit nicht gefordert erleichtert aber die Arbeit mit dem System ungemein. Da jedes Jahr, zum Schuljahresbeginn, neue Schüler in das Kursplanungs- und Kurswahlsystem mit aufgenommen werden müssen wäre der Aufwand einzelne Benutzer hinzuzufügen zu groß und langwierig. Außerdem werden CSV-Dateien auch von gängigen Tabellenkalkulationsprogrammen unterstützt.

Als Vorlage für erste Importversuche diente die aktuelle Schülerliste im CSV-Format. Diese enthielt Daten in der Form:

Code 20: Beispiel einer CSV-Datei mit User Informationen

```
1 'KLASSE','ZUNAME_S','VORNAME_S','GEB_DATUM','RELUNTER'
2 'H7A','Max','Moritz','30.04.1999','RK'
3 'H7A','Moritz','Max','18.07.2000','EV'
4 'H7A','Maria','Maier','27.08.1999','ISL'
5 'H7A','Josef','Ziegler','17.08.2000','KE'
6 'H7A','Metin','Yusuf','05.10.1999','KE'
```

Mit Hilfe dieser Datei konnte ein Parser entworfen werden. Umgesetzt wurde der Parser mit der JAVA-Klasse `StringTokenizer`, um Tokens zu definieren und nach ihnen zu selektieren, und mit der Klasse `BufferedReader` und `InputStreamReader`, um die CSV-Datei überhaupt erst einlesen zu können.

Der Quellcodeausschnitt 21 auf Seite 81 zeigt die ManagedBean-Klasse `importBean` mit der Methode `doImport()`, die den komplettem Parse-Vorgang einer CSV-Datei steuert. Die wichtigsten Zeilen des Quellcodeausschnitts sollen kurz erläutert werden:

Zeile

08: Initialisierung der String-Variablen

23: Beginn des Parsing-Vorgangs: Solange die CSV-Datei weitere Zeilen enthält

25: Initialisierung des StringTokenizers (Zeilen und Angabe des Trennzeichens)

26: Zeilenweise Tokens auswählen, solange weitere Tokens existieren

28: Switch-Case fängt Integer-Wert der Tokens ab und weist die Werte den richtige Strings zu

47: Aufruf der Methode `addUser` der `DatabaseHandler`-Klasse, die als Paramter die ausgelesenen Informationen der CSV-Datei enthält und einen neuen User im System anlegt

Wie unschwer zu erkennen ist, ist der Parser sehr einfach gestrickt. Es reicht anzugeben welche Trennzeichen zwischen den Daten benutzt werden (Obwohl der Namen CSV als Trennzeichen das Komma suggeriert, können als Trennzeichen zumindest auch Semikoli verwendet werden). Weiter ist es ausreichend zu wissen wann die CSV-Datei endet und schlussendlich wie die Reihenfolge, der Werte die ausgelesen werden sollen, ist.

Code 21: CSV-Datei Parser-Methode

```
1  @ManagedBean(name = "importBean")
2  @RequestScoped
3  public class ImportBean implements Serializable {
4    ...
5    public void doImport() {
6
7      // Stringvariablen fuer ausgelesene Daten
8      String line = "";
9      String klasse = ""; // (1)
10     String nname = ""; // (2)
11     String vname = ""; // (3)
12     String geb = ""; // (4)
```

```
13      String konf = ""; // (5)
14      String role = "schueler"; // (6)
15
16      try {
17        BufferedReader reader = new BufferedReader(new
              InputStreamReader(
18             file.getInputStream(), "ISO-8859-1"));
19        StringTokenizer st = null;
20        int lineNumber = 0, tokenNumber = 0;
21
22        line = reader.readLine(); // erste Zeile ueberspringen
23        while ((line = reader.readLine()) != null) {
24          lineNumber++;
25          st = new StringTokenizer(line, ","); // Trennzeichen
26          while (st.hasMoreTokens()) {
27            tokenNumber++;
28            switch (tokenNumber) {
29            case 1:
30              klasse = st.nextToken().replaceAll("'", "");
31            case 2:
32              nname = st.nextToken().replaceAll("'","");
33            case 3:
34              vname = st.nextToken().replaceAll("'","");
35            case 4:
36              geb = st.nextToken().replaceAll("'", "");
37            case 5:
38              konf = st.nextToken().replaceAll("'", "");
39            }
40          }
41          dbh.addUser(klasse, nname, vname, geb, konf, role);
42          tokenNumber = 0;
```

```
43          }
44          reader.close();
45        ...
46      }
47      ...
48  }
```

4.5.2 Festlegen von Usernamen

4.5.3 Generierung von Passwörtern

4.6 Kursverwaltung

Neben dem Anlegen von neuen Benutzern, muss das System auch die Funktionalität besitzen, neue Kurse hinzuzufügen.

Darstellung des Stundenplans:

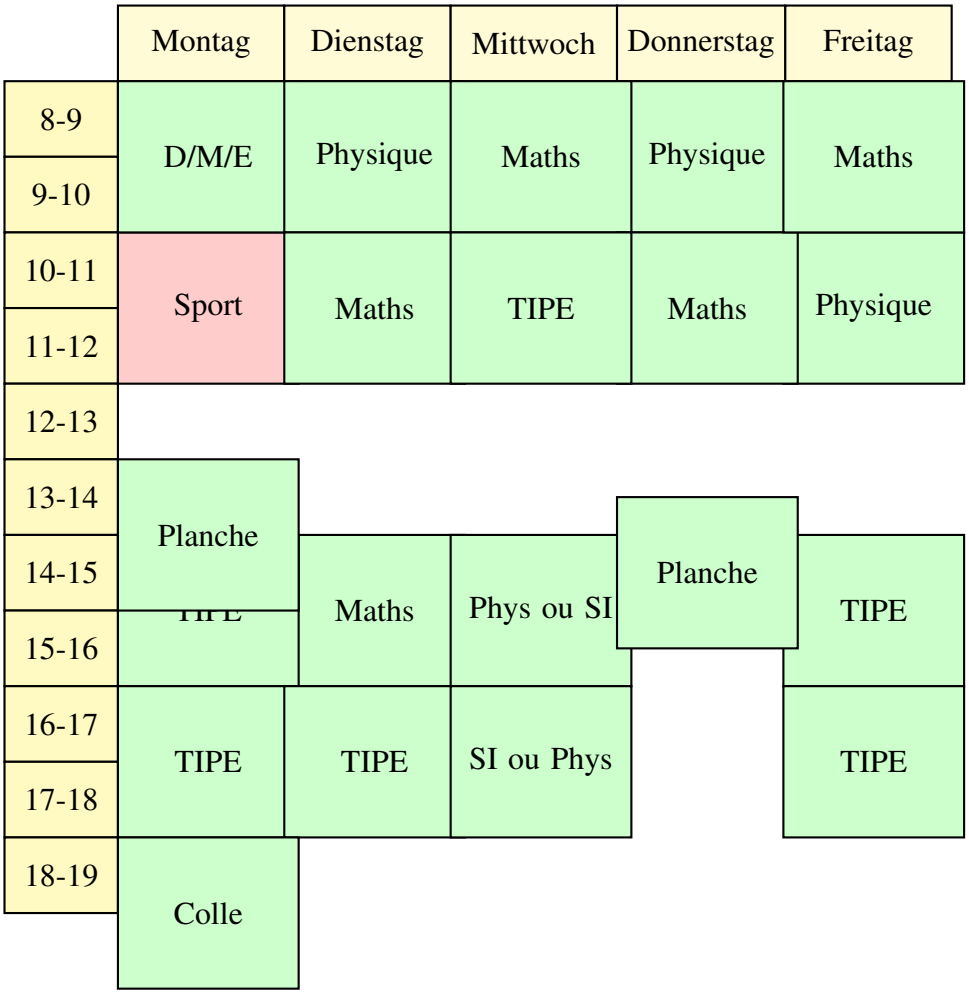


Abbildung 22: Netzwerkstruktur der Schillerschule Aalen

4.6.1 Daten von Kursen

Der Aufbau eines Kurses im System, wie aus dem ER-Modell in Abbildung 13 auf Seite 48 entnommen werden kann, besteht im Grunde aus einer Nummer, einem verantwortlichen Lehrer, einer Beschreibung, einem Termin, Information des dazugehörigen Fächerverbunds und einer Teilnehmeranzahl. Allein mit diesen Information könnte ein Kurs im System dargestelltn werden. Hierbei werden jedoch die restlichen Attribute vernachlässigt. Diese sollen im folgenden näher zur Sprache kommen.

Da die angebotenen Kurse im System nun verschiedenen Abhängigkeiten haben können und eventuell bestimmte Vorraussetzungen erfüllen können müssen, wurde eine Möglichkeit gesucht diese Kurse möglichst einfach im System abzubilden. Einfach bedeutet an dieser Stelle, dass die Darstellung eines Kurses für alle Rollen gleichermaßen zur Verfügung stehen muss und außerdem Interaktionen mit ihnen unterstützt. Hierbei müssen die oben vernachlässigten Attribute beachtet werden.

Attribute die hierbei einer Rolle spielen sind: (Vrgl. ER-Modell in Unterabschnitt 3.4.1 auf Seite 47)

- Sportkurs
- Konfession (derzeit EV, RK und Ethik)
- Pflichtkurse

Sportkurse

Hierbei handelt es sich um ein Attribut, also einer Spalte in der Kurs-Tabelle, die den Datentyp `boolean` besitzt. Beim Anlegen hat der Lehrer oder der Admin die Möglichkeit dieses Attribut des zu erstellenden Kurses zu setzen, wie es die grüne Markierung in Abbildung 23 auf Seite 86 zeigt.

Das anlegen des Sportkurs-Flags in der DB ist trivial und wird daher nicht weiter betrachtet.

Von einer Besonderheit bei Sportkursen kann gesprochen werden wenn zusätzlich die Fächerverbünde und Pflichtkurse mitbetrachtet werden. Für gewöhnlich wird ein Sportkurs dem Fächerverbund Musik-Sport-Gestalten (MSG) zugeordnet. Es kann allerdings vorkommen, dass ein Kurs als Sportkurs angelegt wird, aber keinen Pflichtkurs darstellt. Pflichtkurse werden am Ende dieses Abschnitts noch behandelt.

Beim Anlegen hat der Lehrer oder der Admin die Möglichkeit dieses Attribut des zu erstellenden Kurses zu setzen, wie es die grüne Markierung in Abbildung 23 auf Seite 86 zeigt.

Religionsunterricht

Eine richtige Besonderheit stellt das Anlegen eines Religionskurses dar. Das Anlegen eines Religionskurses (grüne Markierung in Abbildung 23 auf Seite 86) wird über einfach Selectboxen realisiert. Der Inhalt der Selectboxen wird dazu automatisch angelegt. Wie dies gewährleistet wird soll nun näher besprochen werden.

Wie bereits in Abbildung 20 auf Seite 78 gezeigt wurde, wird beim Anlegen eines Users im System, eine Konfessionszugehörigkeit beigefügt. Dieses Feld kann vom Admin beliebig ausgefüllt werden. Dieser beliebige String wird in die Tabelle 'Kurskonfession' eingetragen und wird später, durch die Auswahl beim Anlegen eines Kurses, als Fremdschlüssel in der 'Kurs'-Tabelle gespeichert. Der Inhalt der Selectboxen die zur Verfügung stehen, ist also immer komplette Inhalt der Kurskonfession-Tabelle.

Diese Art der Lösung soll in Zukunft alternative Unterrichte zulassen können.

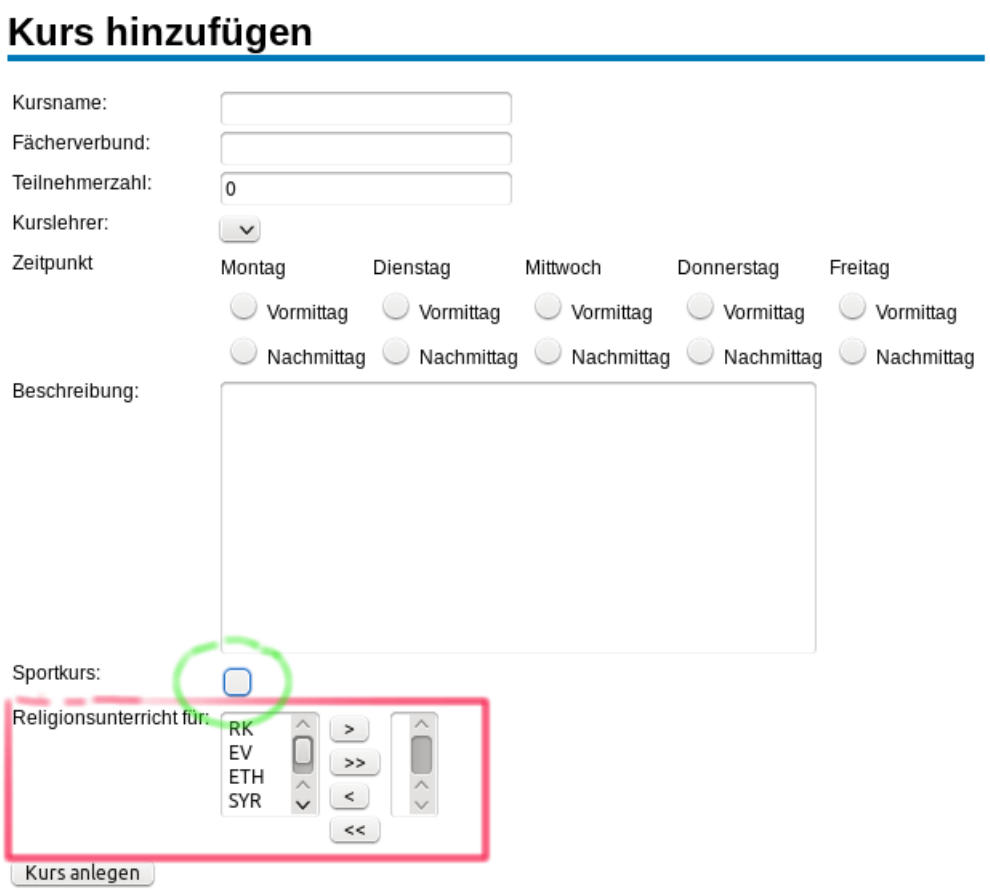


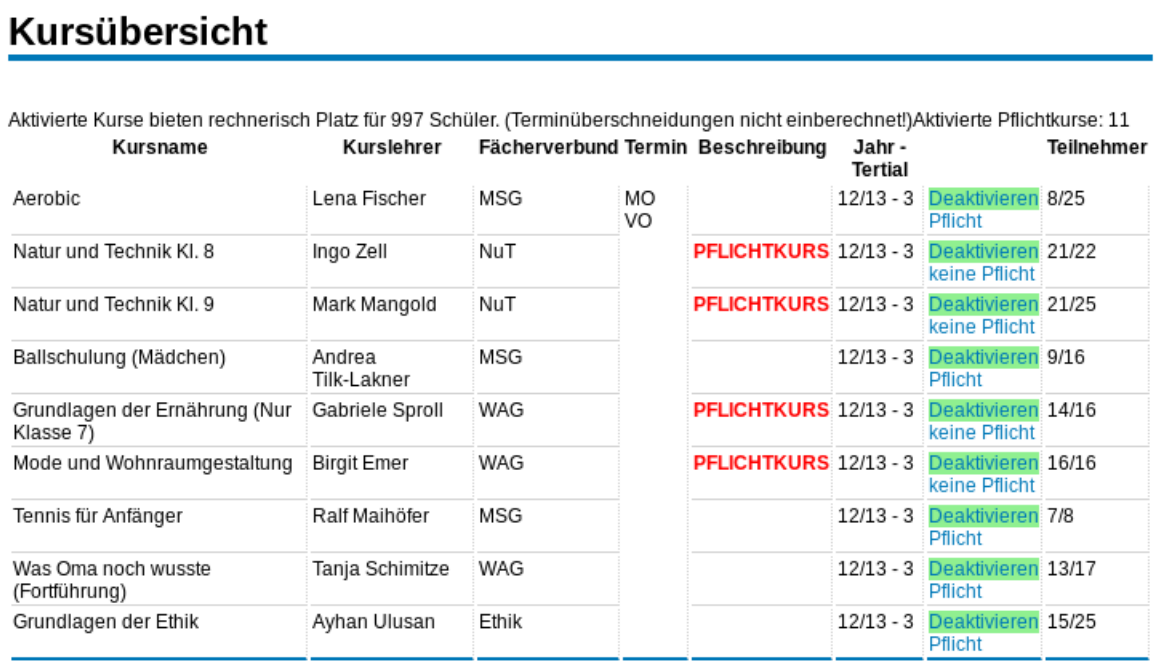
Abbildung 23: KuWaSys: Kurs anlegen

Pflichtkurse

Ähnlich wie bei Sportkursen handelt es sich hierbei um ein Attribut, also ebenfalls einer Spalte in der Kurs-Tabelle, die auch den Datentyp `boolean` besitzt. Im Gegesatz zu Sportkursen wird das Pflichtkurs-Flag allerdings nicht beim Anlegen eines Kurses gesetzt. Es kann nur vom Admin bestimmt werden, ob ein Kurs ein Pflichtkurs ist oder nicht. Festlegen kann er dies in der Kursverwaltung in der Phase der Kursplanung.

Die Abbildung 24 auf Seite 87 zeigt einen Ausschnitt der Kursverwaltung aus Sicht des Admins. Der Admin hat die Möglichkeit Kurse zu aktivieren, also den Schülern

die Kurse zur Kurswahl zur Verfügung zu stellen oder bereits aktivierte Kurse wieder zu deaktivieren. Darüberhinaus legt der Admin die Pflichtkurse fest.



Kursübersicht

Aktivierte Kurse bieten rechnerisch Platz für 997 Schüler. (Terminüberschneidungen nicht einberechnet!)Aktivierte Pflichtkurse: 11

Kursname	Kurslehrer	Fächerverbund	Termin	Beschreibung	Jahr - Tertial		Teilnehmer
Aerobic	Lena Fischer	MSG	MO VO		12/13 - 3	Deaktivieren Pflicht	8/25
Natur und Technik Kl. 8	Ingo Zell	NuT		PFLICHTKURS	12/13 - 3	Deaktivieren keine Pflicht	21/22
Natur und Technik Kl. 9	Mark Mangold	NuT		PFLICHTKURS	12/13 - 3	Deaktivieren keine Pflicht	21/25
Ballschulung (Mädchen)	Andrea Tilk-Lakner	MSG			12/13 - 3	Deaktivieren Pflicht	9/16
Grundlagen der Ernährung (Nur Klasse 7)	Gabriele Sproll	WAG		PFLICHTKURS	12/13 - 3	Deaktivieren keine Pflicht	14/16
Mode und Wohnraumgestaltung	Birgit Emer	WAG		PFLICHTKURS	12/13 - 3	Deaktivieren keine Pflicht	16/16
Tennis für Anfänger	Ralf Maihöfer	MSG			12/13 - 3	Deaktivieren Pflicht	7/8
Was Oma noch wusste (Fortführung)	Tanja Schimitze	WAG			12/13 - 3	Deaktivieren Pflicht	13/17
Grundlagen der Ethik	Ayhan Ulusan	Ethik			12/13 - 3	Deaktivieren Pflicht	15/25

Abbildung 24: KuWaSys: Kurse verwalten

4.7 Systemverwaltung

5 Tests, Fehlervermeidung und Qualitätssicherung

Dieser Abschnitt widmet sich dem Thema der Testverfahren in Softwareprojekten mit dem Schwerpunkt bezogen auf das entwickelte JSF-Projekt 'KuWaSys'. Zuerst sollen die allgemein geltenden Grundlagen angesprochen werden, darauf aufbauend werden die im Projekt verwendeten Testverfahren näher erläutert.

Tests lassen sich in der Software Entwicklung nach ihrer Art und Weise klassifizieren. Auf der einen Seite steht die Konstruktive-Qualitätssicherung auf der anderen Seite die Analytische.

Zu den Konstruktiven-Qualitätssicherungsmaßnahmen zählen Durchführungen während dem und am Entwicklungsprozesses selbst:

- Checklisten
- Programmier-Guidlines
- Templates

Die Analytischen-Qualitätssicherung, zu welchen Tests am Produkt gehören, können in zwei verschiedene Testverfahren eingeteilt werden. Hierbei handelt es sich um statische und dynamische Tests. Verfahren die hierzu eingesetzt werden sind für statische Test:

- Statische Analyse
- Reviews

Im Sinne der statischen Analyse wurden während der Entwicklung folgende Punkte überprüft:

- existieren Klassen bzw. Methoden
- sind die Scopes der ManagedBeans korrekt
- sind ManagedBeans mit dem Scope `Session` und `Application` serialisierbar
- haben ManagedBeans Properties die benötigten Setter-/Getter-Methoden

Und für dynamische Test beispielsweise:

- Blackbox-Testverfahren
- Whitebox-Testverfahren

Die Planung und Strukturierung der Durchführung von Softwaretests, aber auch die Überlegung von Testfällen die am System geprüft werden sollen, stellen einen äußerst wichtigen Prozess der Softwareentwicklung dar. Diese Schritte fließen bestenfalls zu Beginn des Projekts, in der Planungsphase, mit ein. ([**?**], ab Seite 5undzwölfzig)

Als geplante Tests am fertigen System wurden die folgenden vorgesehen, die im Checklisten-Verfahren abgearbeitet wurden:

- Tests bezügliche der Interaktion am System (mit Testpersonen und deren Feedback)
  - prüfen der Übersichtlichkeit aller Benutzergruppen
  - Kontrolle des Seitenaufbaus und der Strukturierung für effizientes Arbeiten
- Testeingaben in Textfelder
  - Tests der Schnittstellen und der übergebenen Parameter nach Datentyp

  - Abklärung von eventuellen Encoding-Problemen

- Funktionalitätstests aller Schaltflächen

  - absichern von korrekten Systemereignissen bzw. -funktionen
  - Kontrolle der übergebenen Werte und Datentypen, um Fehlbelegungen auszuschließen

- Konsistenz der Datenbank

  - Abfragen mit Hinblick auf Eindeutigkeit, bspw. bei `UNIQUE-Constraints` sowie `PRIMARY KEY-` und `FOREIGN KEY-Constraints`
  - Überprüfung der verwendeten Datentypen im System

Im folgenden sollen Tests im Zusammenhang mit der Softwarequalitätssicherung und die Umsetzung im 'KuWaSys' näher betrachtet werden.

Der Stellenwert der Qualitätssicherung von Software hat in der heutigen Zeit einen so großen Stellenwert angenommen, dass es mittlerweile sogar ISO-Normen gibt, genauer gesagt die ISO-Norm 9126 - für Qualität in Software. Diese Norm enthält grundlegende Bestimmungen über Effizienz, Funktionalität, Zuverläsigkeit, Benutzbarkeit, Portabilität und Wartbarkeit. Die genauen Details der Definitionen werden hier nicht weiter erwähnt da diese über das Thema der Projektarbeit hinaus gehen würden. Eine gute und vollständige Zusammenfassung der gesamten Norm ist jedoch unter [de.13a] zu finden.

Abbildung 25: ISO Norm 9126

Grundsätzlich wurden allgemein gültige Richtlinien und Code-Konventionen von JSF im Projekt umgesetzt. Dies erhöht zum eine die Übersichtlichkeit des gesamten Projekts und wirkt sich postiv auf die kooperierende Entwicklung aus. Dadurch werden gleichermaßen qualitätsichernde Maßnahmen ergriffen wie auch Fehleranfälligkeiten minimiert.

Das Benutzen von Templates, welche während der Implementierungsphase eingesetzt wurden, stellt ebenfalls eine Technik dar, die zur Fehlervermeidung beiträgt. Das verwendete Design des 'KuWaSys' musste somit nur einmal definiert und getestet werden und konnte anschließend beliebig oft weiter verwendet werden ohne dabei das Risiko eingehen zu müssen dass das Design inkonsistent wird oder dass sich neue Fehler im Projekt einschleichen könnten.

Um Laufzeittests durchzuführen wurden FacesMessages, über den entsprechenden Import der Klasse `javax.faces.application.FacesMessage` benutzt, die Gleichzeitig die Fehleranalyse mit Hilfe der Konsole erleichtern. Ein Beispiel solcher Messages ist in Quellcodeausschnitt 22 auf Seite 92 dargestellt. Im Falle eines Fehlers würde der ausgeführte `Catch`-Block, über den benutzerdefinierten FacesMessages-Befehl, eine Ausgabe produzieren. Nebenbei sei noch eine weitere Besonderheit in JSF angemerkt: Während der Status 'Development' im Projekt steht, welcher in der `pom.xml` festgelegt wird, werden Fehler für bestimmte Funktionen automatisch ausgegeben. Dies ist vor allem bei einem Release zu beachten und vorher abzuändern.

Code 22: Debugging mit FacesMessages

```
1 try{ ... }
2
3 catch (Exception e) {
4
5   message = new FacesMessage(FacesMessage.SEVERITY_ERROR, "
      Upload␣fehlgeschlagen", null);
6
7 }
8
9 FacesContext.getCurrentInstance().addMessage("csvimport",
    message);
```

Wie in der folgenden Abbildung 26 zu erkennen ist, wird die Message während der Laufzeit ausgeführt, wie bei diesem missglückten Datei-Upload.

Abbildung 26: KuWaSys: Datei-Upload Fehler beim Schüler Importieren

Diese Tests erwiesen sich bei Datenbankabfragen als sehr hilfreich, da nicht erst aufwändige Oberflächen umgesetzt werden müssen um Fehler zu erkennen, sondern Daten sofort auf ihre Richtigkeit hin geprüft werden können. Diese Fehler sind gleich zu Beginn der Implementierung aufgefallen, da es sich hier vor allem um Fehler die während der Modellierungs- beziehungsweise (bzw.) Designphase entstanden sind, handelt. Diese Fehler sind mit FacesMessages relativ schnell auszumachen und ebenso schnell korrigiert. Beispiele solcher Fehler sind:

- falsche Überlegungen zu den Datentypen in der DB
- schlecht definierte Schnittstellen
- Inkonsitenz im Design

Eine weitere Möglichkeit, Fehler in einer Webapplikation zu entdecken ist das benutzen von selbstdefinierten Server-Logs. Hierzu wurde von uns die Klasse `import java.util.logging.Logger` verwendet. Der Quellcodeausschnitt 23 auf Seite 93 zeigt wie beispielsweise ein `Catch`-Block mitgeloggt werden kann.

Code 23: Server-Logging in JSF

```
1 private static Logger logger = Logger.getLogger(ImportBean.
    class
```

```
2       .getCanonicalName());
3 ...
4
5   logger.info("File␣type:␣" + file.getContentType());
6   logger.info("File␣name:␣" + file.getName());
7   logger.info("File␣size:␣" + file.getSize() + "␣bytes");
8
9 ...
10
11 try{ ... }
12
13 catch (Exception e) {
14
15    logger.info("Upload␣fehlgeschlagen\n" + e.getMessage());
16
17 }
```

Da ständig auf einem aktiven System (Corporate Identity) - glossarntwickelt und getestet wurde konnten Tester aus dem Umfeld der Schule für diesen Zweck eingesetzt werden. Dies erwies sich vor allem beim Entwickeln des Oberflächendesigns als Vorteil. Es konnten Wünsche von Lehrern und Schülern direkt berücksichtigt und umgesetzt werden. Hier bekommen besonders Checklisten und Reviews einen hohen Stellenwert. Nur durch regelmäßige gemeinsame Treffen konnten Projekttermine (neu-)definiert werden.

Im Hinblick auf die vorher besprochene ISO-Norm 9126 erfüllt das Kurswahlsystem alle Bestimmungen. Jedoch muss fairnesshalber dazu gesagt werden, dass Dinge wie Effizienz in Software nie mit einem genauen Maß gemessen werden kann, da viele Faktoren ein Rolle spielen. Zum Beispiel müssen bei gewissen Datenstrukturen Laufzeiteinschränkungen in Kauf genommen werden wenn sich dadurch die Darstellung

effizienter umsetzen lässt. Andererseits können natürlich schneller Datenstrukturen oder Algorithmen in einer viel langsameren Darstellung resultieren. Dasselbe gilt für die Zuverläsigkeit. Natürlich ist das Kurswahlsystem so entworfen worden, dass die Erreichbarkeit und Nutzbarkeit jederzeit gegeben ist. Aber auch hier unterliegt das System mehreren außenstehenden Faktoren auf die ein Entwickler niemals Einfluss nehmen kann.

Selbstverständlich können für den Java Server Faces Standard auch Tests implementiert werden. Diese werden für gewöhnlich bei den Dynamischen-Test angesiedelt. Von uns wurden keine spezielle Frameworks zur Realisierung von Tests verwendet. Vollständigkeitshalber soll allerdings eines der interessantesten Vertreter von Testinstrumenten für JSF erwähnt werden:

Hierbei handelt es sich um JSFUnit, welches auf dem bekannten JAVA-Testframework JUnit aufbaut. Tests können hierzu über die JSFUnit-Konsole oder durch den Aufruf einer Testseite ausgeführt werden. Testmöglichkeiten sind Wert- und Zustandsänderungen in ManagedBeans, setzen von Navigationszielen oder über FacesMessages. Eine besondere Art der Tests sind die 'Acrylic Box'-Testings. Diese verbinden Whitebox und Blackbox-Testverfahren und werden bei den dynamischen Tests eingestuft. Nähere Informationen sind unter [jbo13] zu finden.

Zuletzt soll noch hinzugefügt werden, dass auch ausführliche Tests keine Garantie auf eine vollständige Fehlerfreiheit geben. Allerdings helfen Tests die Fehler möglichst gering zu halten und steigern die Qualität der Software um ein gewisses Maß. Obwohl das Projekt ausführlichst getestet wurde kann es dennoch nicht ausgeschlossen werden, dass noch welche existieren.

6 Evaluierung, Fazit und Ausblick

Abschließend soll ein gesamtheitlicher Überblick der Projektarbeit gegeben werden, indem das entwickelte System komplett betrachtet wird. Für ein sinnvolles Fazit sollen zwei wesentliche Punkte angesprochen werden:

1. Beurteilung der Nutzbarkeit und die gesamtheitliche Umsetzung des Projekts
2. Beurteilung der eingesetzten Technologien

6.1 Evaluierung

An oberster Stelle stand das Ziel, die Anforderungen an das System erfolgreich umzusetzen. Das positive Resultat kann der engen Kooperation während der Implementierungsphase und der ausgiebigen Problemabgrenzung, wie grob zu Beginn in Unterabschnitt 1.2 auf Seite 2 beschrieben ist, angerechnet werden. Vor allem die ersten drei bis vier Wochen nach Projektstart dienten dazu, die Ziele zu definieren. Beinahe jede Woche hielten wir mit den verantwortlichen Personen (in Unterabschnitt A.5 auf Seite 5 namentlich aufgeführt) der Schillerschule Aalen ein Projektmeeting ab, in welchem Ergebnisse des Projektfortschritts präsentiert und diskutiert wurden. Ein weiterer wichtiger Punkt, der zu diesem Ergebnis geführt hat, war das entgegengebrachte Vertrauen von Seiten der Schule.

Wie es Abschnitt 4 zu entnehmen ist, wurden alle Punkte der Systemanforderung erfüllt. Darüberhinaus wurden sogar zusätzliche Funktionalitäten implementiert. Hier wäre die Umsetzung der Export-/Importfunktionen von CSV Dateien, Funktionalität die Userdaten betreffen (bspw. Passwort-Neugenerierung) und die komplette Umsetzung der IT-Infrastruktur der Schillerschule genannt. Insofern muss der entstandene

Fortschritt mit dem neuen Kurswahlsystem im Hinblick auf das erste Kursverwaltungstool gesehen werden.

Das größte Manko, der nicht unterstützte Multi-User Betrieb und die Einschränkung der Lauffähigkeit begrenzt auf ein System, wurden beseitigt. Ebenso wurde mehr Wert auf ein gewisses Maß an Selbstverantwortung bei Lehrern aber auch Schülern gelegt. Im Idealfall wird der Admin nur noch zur Konfiguration des Systems und in Problemfällen aktiv. Auch die Verwaltungsassistenz wurde im neuen Tool erfolgreich umgesetzt. So werden Daten die für die Kursplanung wichtig sind vom System ermittelt und den Benutzern zur Verfügung gestellt. Ebenso wurde der entwickelte Arbeitsablauf für Lehrer und Schüler umgesetzt und die Kommunikation untereinander dadruch verbessert indem Probleme Systembedingt abgefangen werden können.

Als dritter und letzter Verbesserungsschritt ist die neugestaltung der Oberfläche zu sehen. Diese wurden mit Hilfe von bereits erprobten und bewährten Gestaltungselementen der Webentwicklung, die aus dem Gebiet der Mensch-Computer-Interaktion (MCI) bekannt sind, umgesetzt ([Dah08], 256 ff.) .

Die eindeutige Abgrenzung von Menü, Informationen und Arbeitsbereich sind gleichermaßen umgesetzt wie die Darstellung aller Systemrelevanter Daten wie es in der Anforderung gewünscht war.

Zukünftige Erweiterungen sind im System vorgesehen. Aufgrund der Modularität die sich strikt durch das ganze System zieht werden diese auch realisierbar sein.

Realisierbare Erweiterungen könnten sein:

- eine Änderungsinformationsanzeige für Admins bei bestimmten Systemereignissen, sodass ohne fundierte IT-Kenntnisse, Fehler an Server und Datenbank frühzeitig erkannt werden können

- eine interne Kommunikationsplattform für den Austausch von Informationen zwischen Lehrern und Administrator und zwischen Schülern - und Lehrern
- Erweiterungen wie Kalenderfunktionen, Stundenplan-Generierung,....

Manche der aufgeführten Vorschläge werden bereits im Momentanen Stand der Entwicklung, ansatzweise umgesetzt. Aufgrund der Fülle und Diversität der Art an Anforderungen die zu Beginn an das Projekt festgehalten wurden fanden diese jedoch in der Projektarbeit bedauerlicherweise fast keinen Platz.

6.2 Fazit und Ausblick

Es hat sich gezeigt, dass JSF ein gut definierter Webentwicklungsstandard geworden ist. Aufgrund der Einfachheit der Handhabung kommen Anfänger der Webentwicklung aber auch Fortgeschrittene gut damit zurecht und voll auf ihre Kosten. Je nach Komplexität der Interaktion mit dem System sind JAVA Kenntnisse erfoderlich, in jedem Falle aber sinnvoll. Eine Entwicklung die ihren Schwerpunkt auf das Design oder generell die Darstellung legt benötigt nur sehr wenige Kenntnisse im Bereich JAVA dafür aber Kenntnisse einer Seitendeklarationssprache wie HTML oder JSP. Sollen aber, wie es in dieser Projektarbeit der Fall war, Daten aus einer Datenbank verändert oder in eine Datenbank geschrieben werden sind fundierte Kenntnisse der Softwareentwicklung mit JAVA notwendig.

Zur verwendeten Datenbank lassen sich keine besonders wichtigen oder bahnbrechenden Aussagen machen. PostgreSQL hat sich bereits über Jahre etabliert und wurde stetig weiter verbessert. Die Verwendung für unerfahrene Nutzer stellt nach einer kurzen Einarbeitung kein größeres Problem dar und kann somit nur weiterempfohlen werden.

Die Entwicklungsunterstützung mit Maven ist bei Projekten dieser Art eine sehr große

Hilfe. Ohne eine Verwaltungshilfe der Bibliotheken, Pakete oder der Projektverweise kann ein Projekt in dieser Größenordnung schnell in einen Bereich kommen, in welchem plötzlich der Verwaltungsaufwand steigt und schnell zur Hauptaufgabe des Entwicklers wird. Der Fokus schweift immer öfters vom Projekt ab. Das Resultat davon ist:

- ein höherem Zeitaufwand für das Beheben von entstandenen Problemen durch die Verletzung von Abhängigkeiten
- mehr Fehler im Programmcode der wiederrum auch zu längeren Testphasen führt
- teilweise leidet die Qualität des entstanden Codes

Für viele Entwickler (oder Projektleiter) ist dies ein stark vernachlässigtes Thema. Ohne eine genaue Planung und ohne konkrete Vorstellungen lässt sich jedoch ein Softwareprojekt, egal welcher Größe, meist nie korrekt realisieren. Dies sind alles Gründe die dafür gesorgt haben, dass das Projekt unter zu Hilfenahme des Development-Tools Maven umgesetzt wurde. Die Softwareentwicklung bleibt im Vordergrund.

Rückblickend kann dem Projekt ein voller Erfolg zugeschrieben werden. Funktional genügt das System den Anforderungen und sogar darüber hinaus. Der zeitliche Aufwand war sehr hoch, aber im Rahmen des Möglichen, sodass nicht von 'zu viel' gesprochen werden kann. Zeitliche Einbußen musste dennoch hingenommen werden. So stellten vor allem das Testen und die Integration des Systems in die Infratruktur der Schule einen langwierigen Prozess dar und sprengte den geplanten Rahmen. Letztenendes sorgten Dinge wie die Anleitung, die technische Dokumentation und diese Ausarbeitung der Projektarbeit für den nötigen Abschluss des Projekts.

A Anhang

A.1 Inhalte der CD

Folgende Inhalte sind auf der abgegebenen CD zu finden:

- Kopie der Entwicklungsumgebung Eclipse 10.04 Juno mit allen Plugins
- virtuelles Abbild des konfigurierten Servers
- diese Ausarbeitung
- Dokumentation in JAVAdoc
- Benutzerhandbuch für das 'KuWaSys'
- Kopien der verwendeten bzw. zitierten Websites

A.2 Konfiguration und Installation

A.2.1 Maven

Maven Befehle listings

A.2.2 Apache Tomcat 7

Hier soll in einer kurzen Darstellung die Administration des Apache Tomcat 7 und dessen wichtige Dateien und Verzeichnisse

Apache Tomat7 Befehle Listings

Verzeichnisse

A.3 Zugangsdaten für den Server und das System

Server Zugangsdaten

Rolle	Username	Passwort
Admin	ijcy	12kuwasys34

Tabelle 9: Server Zugangsdaten

Datenbank Zugangsdaten

Rolle	Username	Passwort
Admin	ijcy	12kuwasys34
Admin	postgres	12kuwasys34

Tabelle 10: Datenbank Zugangsdaten

A.4 Gesetzesauszüge

Auszug aus dem TMG:

§ 5 Allgemeine Informationspflichten

(1) Diensteanbieter haben für geschäftsmäßige, in der Regel gegen Entgelt angebotene Telemedien folgende Informationen leicht erkennbar, unmittelbar erreichbar und ständig verfügbar zu halten:

1. den Namen und die Anschrift, unter der sie niedergelassen sind, bei juristischen Personen zusätzlich die Rechtsform, den Vertretungsberechtigten und, sofern Angaben über das Kapital der Gesellschaft gemacht werden, das Stamm- oder Grundkapital sowie, wenn nicht alle in Geld zu leistenden Einlagen eingezahlt sind, der Gesamtbetrag der ausstehenden Einlagen,
2. Angaben, die eine schnelle elektronische Kontaktaufnahme und unmittelbare Kommunikation mit ihnen ermöglichen, einschließlich der Adresse der elektronischen Post,
3. soweit der Dienst im Rahmen einer Tätigkeit angeboten oder erbracht wird, die der behördlichen Zulassung bedarf, Angaben zur zuständigen Aufsichtsbehörde,
4. das Handelsregister, Vereinsregister, Partnerschaftsregister oder Genossenschaftsregister, in das sie eingetragen sind, und die entsprechende Registernummer,
5. soweit der Dienst in Ausübung eines Berufs im Sinne von Artikel 1 Buchstabe d der Richtlinie 89/48/EWG des Rates vom 21. Dezember 1988 über eine allgemeine Regelung zur Anerkennung der Hochschuldiplome, die eine mindestens dreijährige Berufsausbildung abschließen (ABl. EG Nr. L 19 S. 16), oder im Sinne von Artikel 1 Buchstabe f der Richtlinie 92/51/EWG des Rates vom 18. Juni 1992 über eine zweite allgemeine Regelung zur Anerkennung beruflicher Befähigungsnachweise in Ergänzung zur Richtlinie 89/48/EWG (ABl. EG Nr. L 209 S. 25, 1995 Nr. L 17 S. 20), zuletzt geändert durch die Richtlinie 97/38/EG der Kommission vom 20. Juni 1997 (ABl. EG Nr. L 184 S. 31), angeboten oder erbracht wird, Angaben über
   a) die Kammer, welcher die Diensteanbieter angehören,
   b) die gesetzliche Berufsbezeichnung und den Staat, in dem die Berufsbezeichnung verliehen worden ist,
   c) die Bezeichnung der berufsrechtlichen Regelungen und dazu, wie die-

se zugänglich sind,

6. in Fällen, in denen sie eine Umsatzsteueridentifikationsnummer nach § 27a des Umsatzsteuergesetzes oder eine Wirtschafts-Identifikationsnummer nach § 139c der Abgabenordnung besitzen, die Angabe dieser Nummer,
7. bei Aktiengesellschaften, Kommanditgesellschaften auf Aktien und Gesellschaften mit beschränkter Haftung, die sich in Abwicklung oder Liquidation befinden, die Angabe hierüber.

(2) Weitergehende Informationspflichten nach anderen Rechtsvorschriften bleiben unberührt.

Auszug aus dem RstV:

§ 55 - Informationspflichten und Informationsrechte

(1) Anbieter von Telemedien, die nicht ausschließlich persönlichen oder familiären Zwecken dienen, haben folgende Informationen leicht erkennbar, unmittelbar erreichbar und ständig verfügbar zu halten:

1. Namen und Anschrift sowie
2. bei juristischen Personen auch Namen und Anschrift des Vertretungsberechtigten.

(2) Anbieter von Telemedien mit journalistisch-redaktionell gestalteten Angeboten, in denen insbesondere vollständig oder teilweise Inhalte periodischer Druckerzeugnisse in Text oder Bild wiedergegeben werden, haben zusätzlich zu den Angaben nach §§ 5 und 6 des Telemediengesetzes einen Verantwortlichen mit Angabe des Namens und der Anschrift zu benennen. Werden mehrere Verantwortliche benannt, so ist kenntlich zu machen, für welchen Teil des Dienstes der jeweils Benannte verantwortlich ist. Als Verantwortlicher darf nur benannt werden, wer

1. seinen ständigen Aufenthalt im Inland hat,
2. nicht infolge Richterspruchs die Fähigkeit zur Bekleidung öffentlicher Ämter verloren hat,
3. voll geschäftsfähig ist und
4. unbeschränkt strafrechtlich verfolgt werden kann.

(3) Für Anbieter von Telemedien nach Absatz 2 Satz 1 gilt § 9 a entsprechend.

A.5 Verantwortliche Personen

Ralf Meiser - Konrektor

Björn Bolch -

Alexander Neugebauer - IT Systembeauftragter der Schillerschule Aalen

B Akronyme

JSF Java Server Faces

JSP Java Server Pages

POM Project Object Model

IDE Integrated Development Environment

Webapp Webapplikation

UML Unified Modelling Language

UI User Interface

WWW World Wide Web

DMZ Demilitarized Zone

ER-Modell Entity-Relationship Modell

MVC Model-View-Controller

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

JDK Java Development Kit

XML Extensible Markup Language

API Application Programming Interface

POJO Plain Old Java Object

DB Datenbank

DBMS Datenbank Managment System

OOM Objektorientierte Modellierung

UEL Unified Expression Language

UC-Diagramm Use-Case-Diagramm

UC Use-Case

SLES SuSE Linux Enterprise Server

IT Informationstechnik

TMG Telemediengesetz

RstV Rundfunkstaatsvertrages

CSS Cascading Style Sheets

DBCP Database Connection Pool

MSG Musik-Sport-Gestalten

WAG Musik-Sport-Gestalten

WZG Welt-Zeit-Gesellschaft

MNT Mensch-Technik-Natur

CSV Comma Separated Values

VDL View Declaration Language

MCI Mensch-Computer-Interaktion

SVN Subversion

P2P Peer-to-Peer

CI Corporate Identity

C Glossar

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Literatur

[aal13] AALEN.DE schillerschule: *Portrait der Schillerschule - Schillerschule Website*. Version: 2013. http://129.143.227.74/index.php?option=com_content&view=article&id=32&Itemid=3, Abruf: 08. März 2013

[AB03] ANATOL BADACH, Matthias S. Sebastian Rieger R. Sebastian Rieger: *Web-Technologien*. München : Carl Hanser Verlag, 2003

[Bal10] BALZERT, Heide: *UML 2 kompakt*. Heidelberg : Spektrum akademischer Verlag, 2010

[bw.13] BW.DE lehrerfortbildung (Hrsg.): *paedML Novell Musterlösung - LBW Website*. Version: 2013. http://lehrerfortbildung-bw.de/netz/muster/novell/, Abruf: 05. April 2013

[Che76] CHEN, Peter Pin-Shan: The Entity-Relationship Model - Toward a Unified View of Data. In: *International Conference on Very Large DataB ases*, 1976

[Dah08] DAHM, Markus: *Grundlagen der Mensch-Computer-Interaktion*. München : Oldenbourg Wissenschaftsverlag, 2008

[de.13a] DE.WIKIPEDIA.ORG (Hrsg.): *ISO Norm 9126 - Wikipedia Website*. Version: 2013. http://de.wikipedia.org/wiki/ISO/IEC_9126, Abruf: 04. Mai 2013

[de.13b] DE.WIKIPEDIA.ORG (Hrsg.): *Java Server Faces Artikel - Wikipedia Website*. Version: 2013. http://de.wikipedia.org/wiki/JavaServer_Faces, Abruf: 19. März 2013

[ecl13] ECLIPSE.ORG (Hrsg.): *Eclipse Project Website - Maven Integration for WTP*. Version: 2013. http://www.eclipse.org/m2e-wtp/, Abruf: 05. April 2013

[jbo13] JBOSS.ORG (Hrsg.): *JSFUnit - JBoss Inc. JBoss Community Website*. Version: 2013. http://http://www.jboss.org/jsfunit/, Abruf: 04. Mai 2013

[jsf13a] JSFATWORK.IRIAN.AT (Hrsg.): *Lebenszyklus in JSF Anwendungen - Irian Webbook*. Version: 2013. http://jsfatwork.irian.at/book_de/jsf.html#!idx:/jsf.html, Abruf: 11. März 2013

[jsf13b] JSFATWORK.IRIAN.AT (Hrsg.): *Model View Controller in JSF Anwendungen - Irian Webbook*. Version: 2013. http://jsfatwork.irian.at/semistatic/introduction_printpreview.html, Abruf: 11. März 2013

[mav13] MAVEN.APACHE.ORG (Hrsg.): *Introduction to the dependency managment- Apache Software Foundation Website*. Version: 2013. http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html, Abruf: 13. Mai 2013

[PM07] PETER MAHLMANN, Christian S.: *Peer-to-Peer Netzwerke*. Berlin Heidelberg : Springer-Verlag, 2007

[pos13a] POSTGRESQL.ORG (Hrsg.): *PostgreSQL About - PostgreSQL Global Development Group Website*. Version: 2013. http://www.postgresql.org/about/, Abruf: 11. März 2013

[pos13b] POSTGRESQL.ORG (Hrsg.): *PostgreSQL Download - PostgreSQL Global Development Group Website*. Version: 2013. http://www.postgresql.org/download/, Abruf: 11. März 2013

[scm13] SCM.COM git (Hrsg.): *Git Dokumentation - Git Website*. Version: 2013. http://git-scm.com/documentation, Abruf: 15. Mai 2013

[tom13] TOMCAT.APACHE.ORG (Hrsg.): *Database Connection Pool - Apache Software Foundation Website*. Version: 2013. http://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html#Introduction, Abruf: 28. April 2013

[Vos08] VOSSEN, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. München : Oldenbourg Wissenschaftsverlag, 2008