Praktikumsverwaltung

Projektdokumentation zum Softwareentwicklungsprojekt

(Entwicklerdokumentation)

Lehrveranstaltung „Software Engineering I / II“

18. Juli 2014

Entwickler: Rick Hermenau, Ivo Beier, Jakob Heltzig, Sepp Härtel

Auftraggeber: Prof. Dr.-Ing. Arnold Beck

Diplomstudiengang Medieninformatik
Hochschule für Technik und Wirtschaft Dresden

Zusammenfassung

Zur Verwaltung der Studenten im Praktikumssemester oder Projektsemester wurde ein System gefordert, welches sich intuitiv bedienen lässt und die Funktionen des Vorgängersystems ersetzt und erweitert.

Inhaltsverzeichnis

1 Einleitung

Die Aufgabe bestand darin, für den Praktikumsverantwortlichen der HTW–Dresden eine Anwendung zu entwerfen, mit der sich das Praktikumssemester der Studenten verwalten lässt (detaillierte Anforderungen sind im Pflichtenheft einzusehen). Die Entwicklung hat den Status erreicht, dass das Programm in der Praxis verwendet werden kann. Durch die Modularisierung und Dokumentation sollte es nach kurzer Einarbeitungszeit möglich sein Erweiterung und Anpassungen vorzunehmen.

2 Projektmanagement

2.1 Vorgegebener Zeitablauf

Vorlage Pflichtenheft	31.01.2014
Bearbeitung Analyse/Entwurf:	01.03.2014 – 21.07.2014
Klassendiagramme, Komponentendiagramme, Paketstruktur; Verteilungsdiagramme Festlegung der Rollenverteilung	01.03.2014 – 21.07.2014
Durchführung der Implementierung User Interface Prototyping Bereitstellung API (Javadoc) Testung (JUnit)	01.03.2014 – 21.07.2014

Abgabe der Projektdokumentation	23.06.2014
Vorbereitung Präsentation¨	25.06.2014

Präsentation	26.06.2014
Abgabe aller Dokumente	21.07.2014

2.2 Ressourcenplanung und Organisation

Das Projekt wurde von vier Studenten in einem Studiensemester angefertigt. Wichtige Projektentscheidungen wurden im Kollektiv getroffen. Nach der Planungsphase bildeten sich relativ schnell „Spezialisten“ heraus. Ein kleiner Kern kümmerte sich um die Architektur und experimentierte mit verschiedenen Entwürfen, während parallel andere Funktionalitäten entwickelt wurden.

Folgende Rollen wurden verteilt:

Rolle	**Mitarbeiter**	**Aufgaben**
Projektleiter	Rick Hermenau	Zeitmanagement, Aufgabenverteilung
Usability Manager	Rick Hermenau, Jakob Heltzig, Iwo Bayer, Sepp Härtel	Diskussion über GUI Gestaltung und Verknüpfungen
Architekt	Rick Hermenau , Sepp Härtel	Anwendung von Design Patterns, Prüfen von Architektur auf Modularität
Programmierer	Rick Hermenau, Iwo Bayer, Jakob Heltzig, Sepp Härtel	
Tester	Rick Hermenau, Iwo Bayer	Prüfen ob geforderte Funktionalität gewährleistet ist

2.3 Werkzeugunterstützung

2.3.1 Managementwerkzeuge

Als Managementwerkzeuge wurden eingesetzt:

- Git und SVN für das Code-Management
- Evernote bzw. OneNote für das Management der Aufgaben bzw. Rollenverteilung

2.3.2 Softwareentwicklungswerkzeuge

Als Softwareentwicklungswerkzeuge wurden eingesetzt:

- JDK 1.7
- Eclipse-Kepler
- Netbeans

3 Pflichtenheft

An dieser Stelle wird auf das gesonderte Dokument „Pflichtenheft“ verwiesen.

4 Anforderungsanalyse und Entwurf

4.1 Anwendungsfallanalyse

Laut Anforderungsanalyse des Pflichtenheftes gibt es im System nur einen Akteur, den Praktikumsverantwortlichen.

Im Pflichtenheft wurden für jeden der Bereiche Student, Firmen, Betreuer und Verträge funktionale Anforderungen, wie anlegen, suchen, bearbeiten, usw. definiert, die wie folgt umgesetzt wurden.

Funktionen in allen Bereichen:

Abbildung 1: Suche über Suchfeld

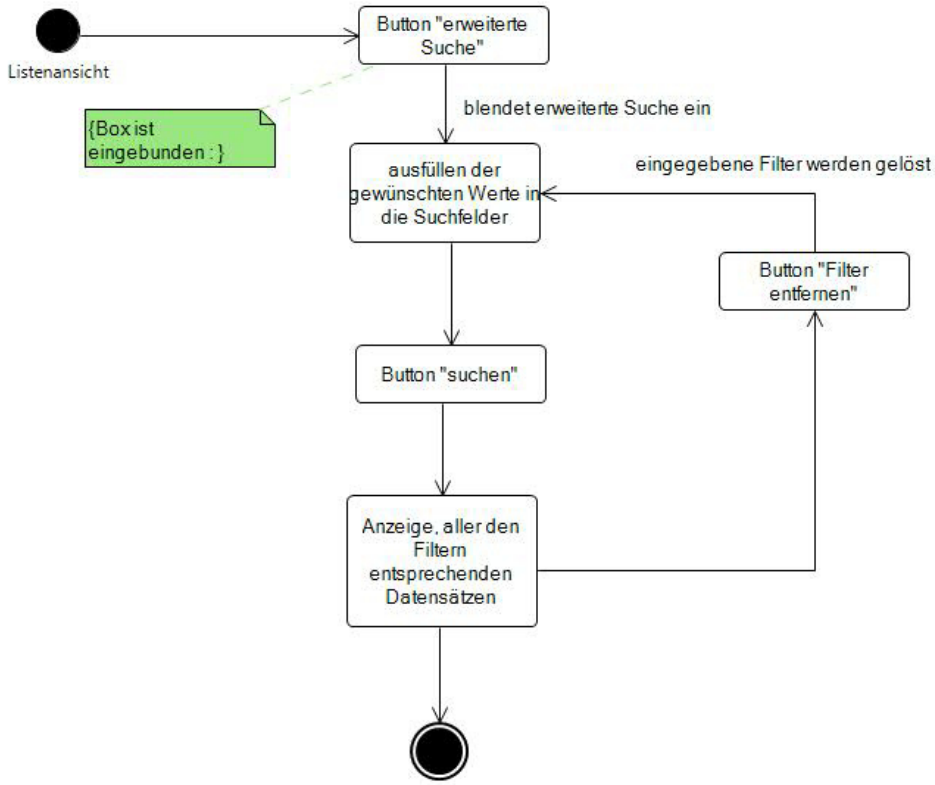


Abbildung 2: Erweiterte Suche

Funktionen im Bereich Student:

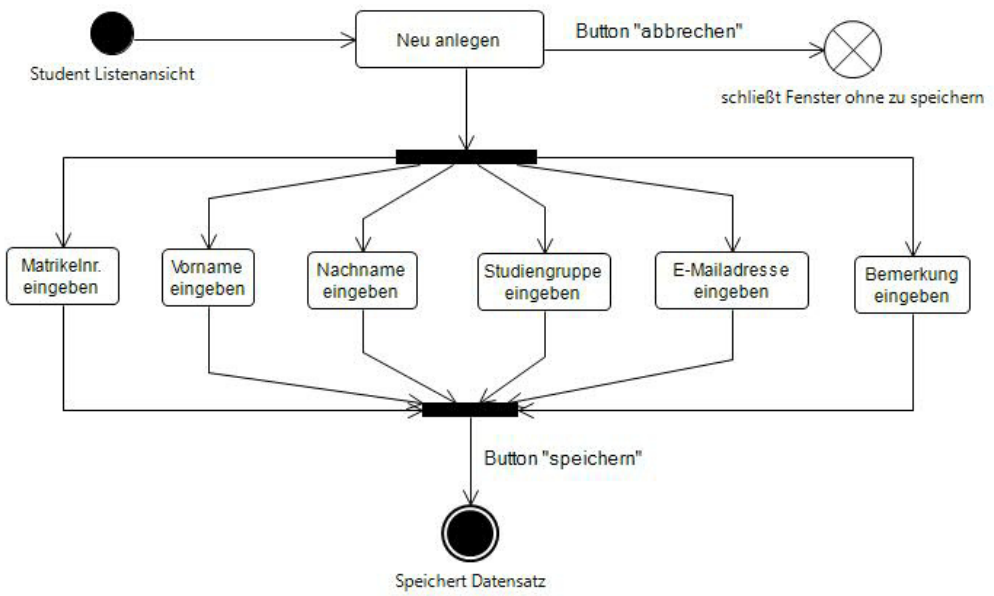


Abbildung 3: Student anlegen

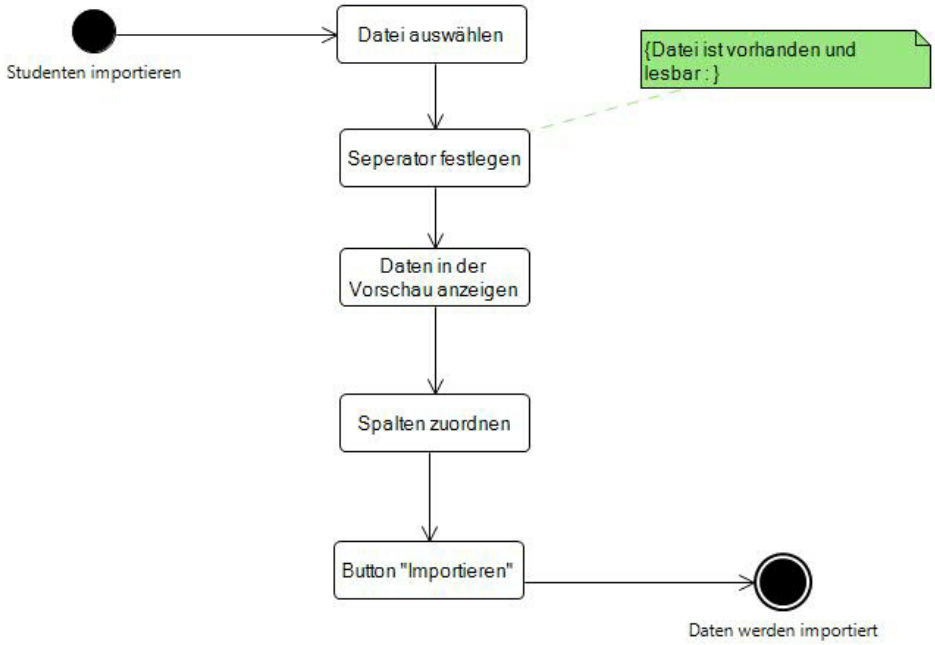


Abbildung 4: Import von Studentendaten aus einer Datei.

Funktionen im Bereich Betreuer:

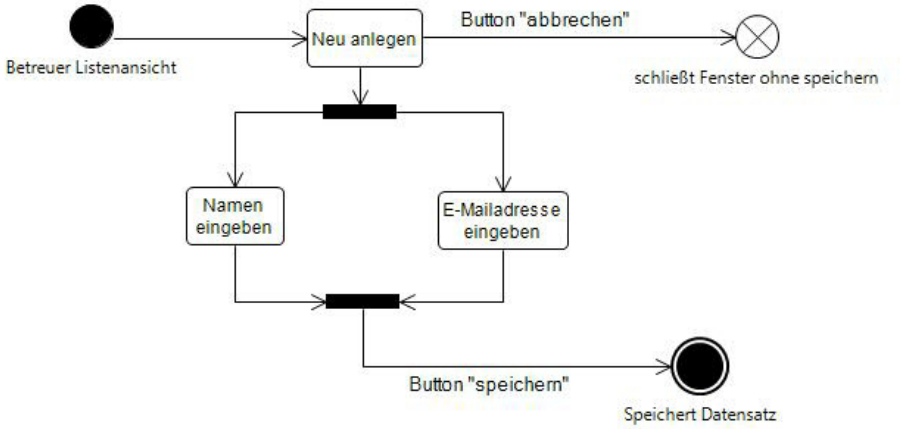


Abbildung 5: Betreuer anlegen

Funktionen im Bereich Firma:

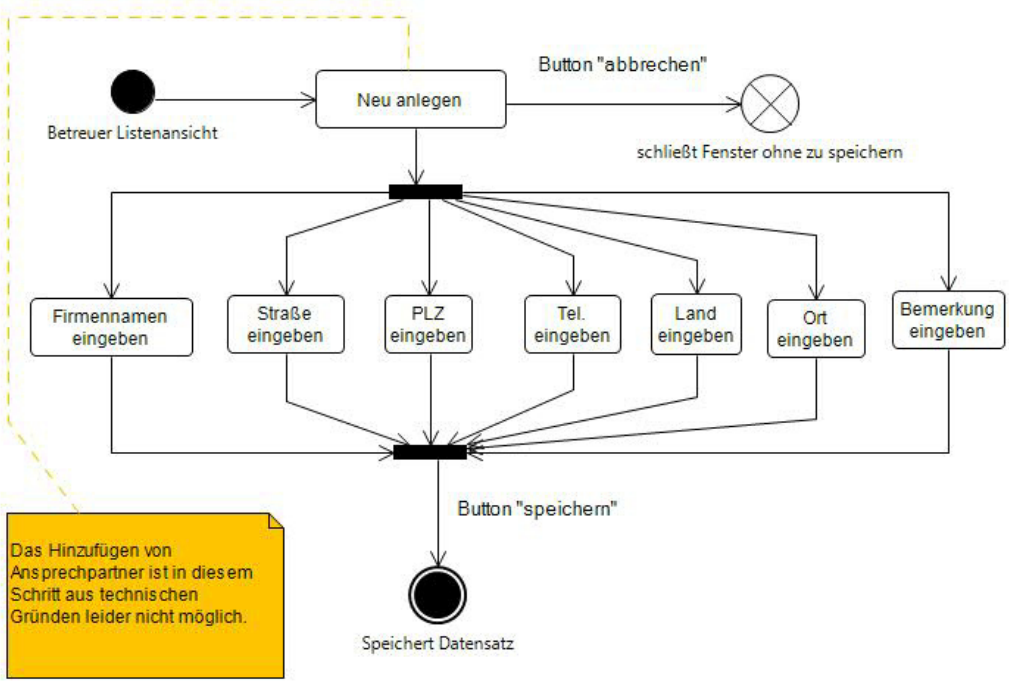


Abbildung 6: Firma anlegen

Funktionen im Bereich Verträge:

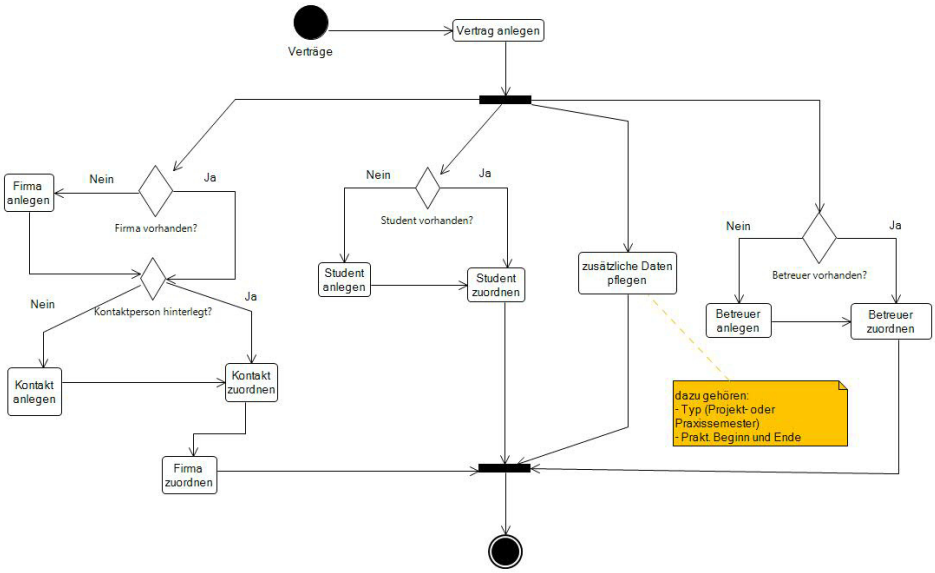


Abbildung 7: Vertrag anlegen

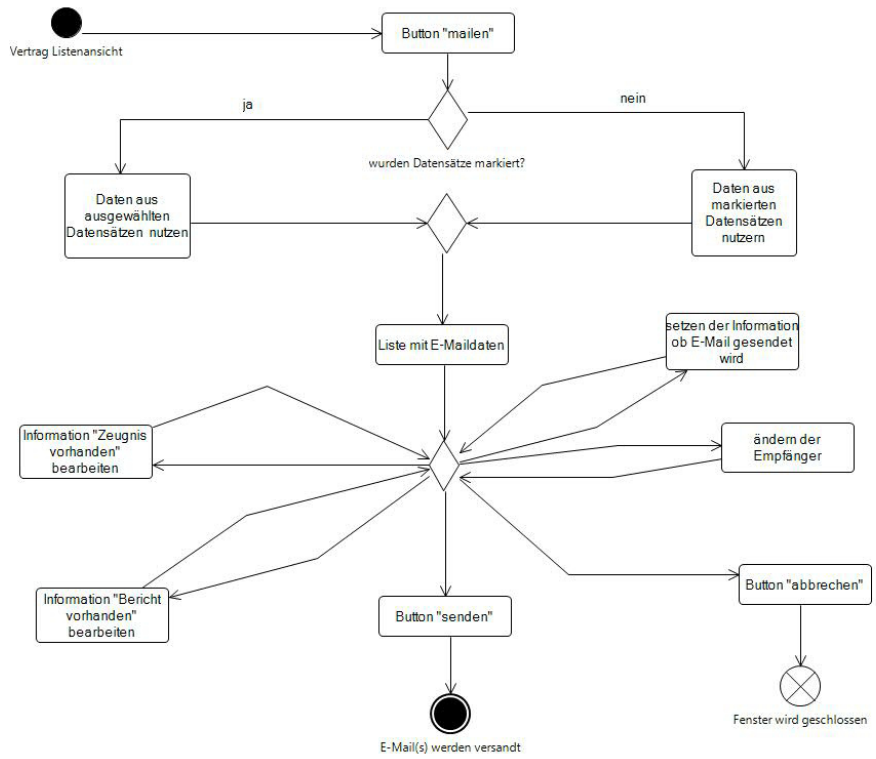


Abbildung 8: E-Mails versenden

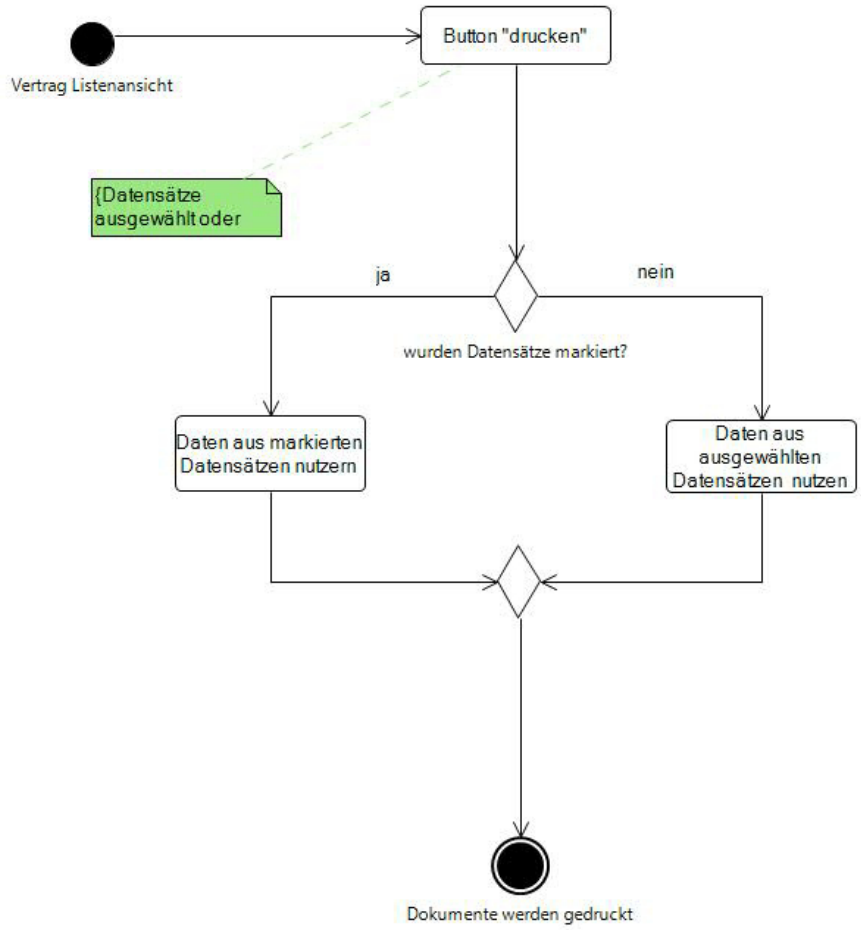


Abbildung 9: Dokument drucken

4.2 Problembereichsanalyse

Der Problembereich des Projektes „Praktikumsverwaltung“ umfasst den Informationsfluss und die Informationshaltung von Verträgen des Praktika- bzw. Projektsemesters von Studenten.

Ein Vertrag besteht aus Informationen zum Vertrag[1] und der Zuordnung einer Firma, eines Studenten und eines Betreuers.

Die Daten liegen in tabellarischer Form vor, welche in ihrer Struktur nicht verändert werden sollte.

Analog dazu werden die detaillierten Informationen zu Firmen, Studenten und Betreuern ebenfalls in tabellarischer Form in der Datenbank gehalten.

Auf dieser Grundlage und der Forderung die Daten übersichtlich darzustellen, haben wir uns für eine tabellarische Ansicht für die Auflistung der Datensätze entschieden.

Die Bearbeitung der einzelnen Datensätze ist in dieser Ansicht jedoch ungeeignet. Eine bessere Übersicht und einen höheren Freiheitsgrad der Darstellung der Informationen erreichten wir mittels einer Karteikartendarstellung.

Die Struktur der Datenmodelle ist in jeder Darstellung identisch. Alle im Projekt genutzten Datenmodelle sind tabellarisch aufgebaut und beinhalten die Ergebnisse einer Sql-Datenabfrage.

D.h. das es nur eine Definition der Datenhaltung benötigt, um alle Informationen des derzeitigen Datenbestandes abzubilden.

Wir haben uns klar gegen die Einführung von Abstraktionen der verschiedenen Daten (Student, Vertrag, Firma, Betreuer) entschieden, mit der Begründung, dass die Datenhaltung der Datenbank und die hauptsächliche Darstellung auf einer Tabellenstruktur basiert und die Daten immer in dieser Form von der Datenbank geliefert werden.

[1] Vertragstyp (Ausland, Inland), Beginn und Ende des Praktikums, Vorliegen von Bericht, Zeugnis, Empfehlung und Erfolg des Praktikums.

4.3 Stand der Wissenschaft und Technik

Da wir auf keinem Softwarestand aufbauen und alles von Grund auf neu entwickelt haben, war es für uns nur wichtig, den Problembereich und die Geschäftsprozesse zu kennen.

4.4 Entwurf der Systemarchitektur

Um eine modulare Softwarearchitektur zu erhalten und zudem eine Trennung von Funktion und Design zu gewährleisten, wurde das Architekturmuster „Model-View-Controller“[1] angewandt.

In unserer Umsetzung übernimmt der Controller die Zusammensetzung der Anzeige, die Reaktion auf Ereignisse dieser und die Definition der Daten im Model (er beschreibt die Sql-Abfrage).

Die View dient als Container für *BasicBox* Elemente, diese wiederum bringen die Daten des Models zur Anzeige. Der Datenbezug wird über den Controller realisiert.

Das Model hält die Daten der Sql-Abfrage und bietet Methoden zum Manipulieren und ausliefern der Daten an die View. Die eigentliche Datenbankschnittstelle ist in der *Database* Klasse eingebettet. Die *Database* Klasse ist nach dem „Singleton[2]“-Architekturmuster implementiert und nutzt zusätzlich die Eigenschaften des „Observer[3]“-Musters um alle Models (welche das Interface *Observer* implementieren) bei einer Änderung der Daten zu informieren.

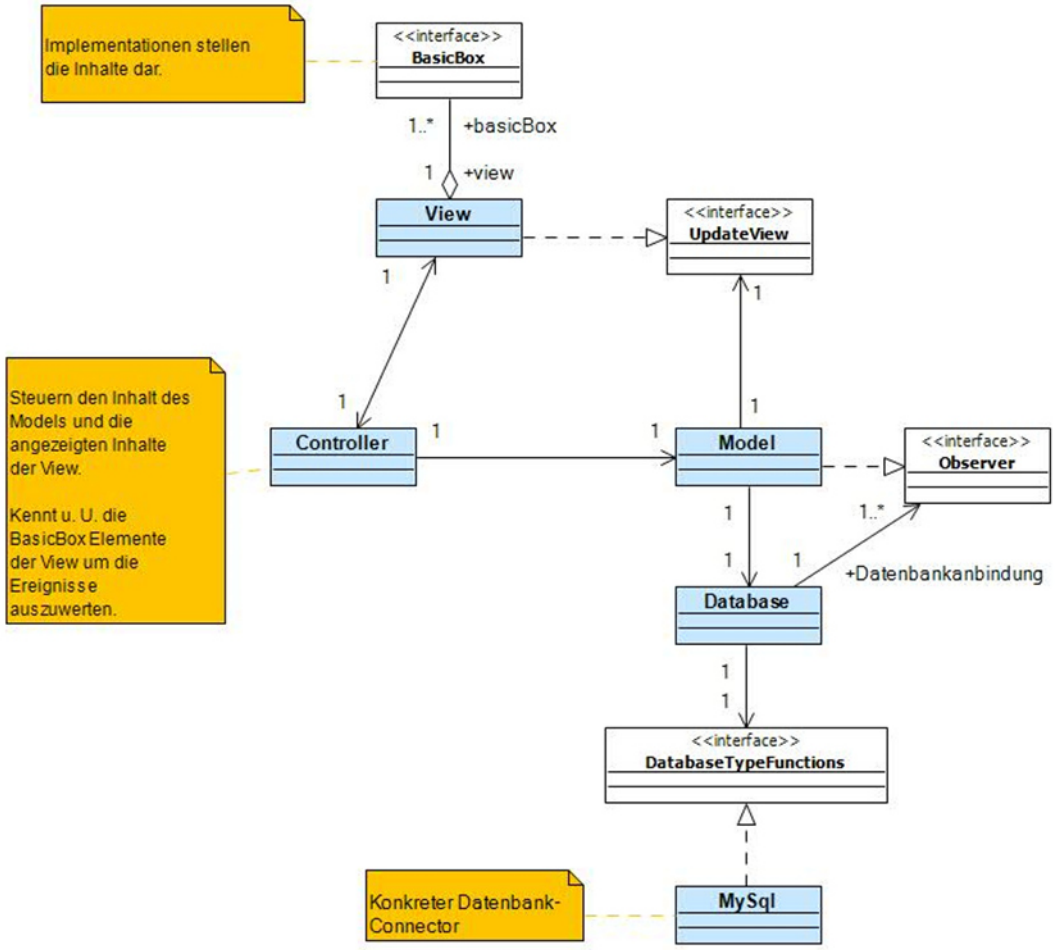


Abbildung 10: Grobe Darstellung der Systemarchitektur

[1] http://de.wikipedia.org/wiki/Model_View_Controller
[2] http://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster)
[3] http://de.wikipedia.org/wiki/Beobachter_(Entwurfsmuster)

Um alle geforderten Funktionen umzusetzen, benutzen wir externe APIs.

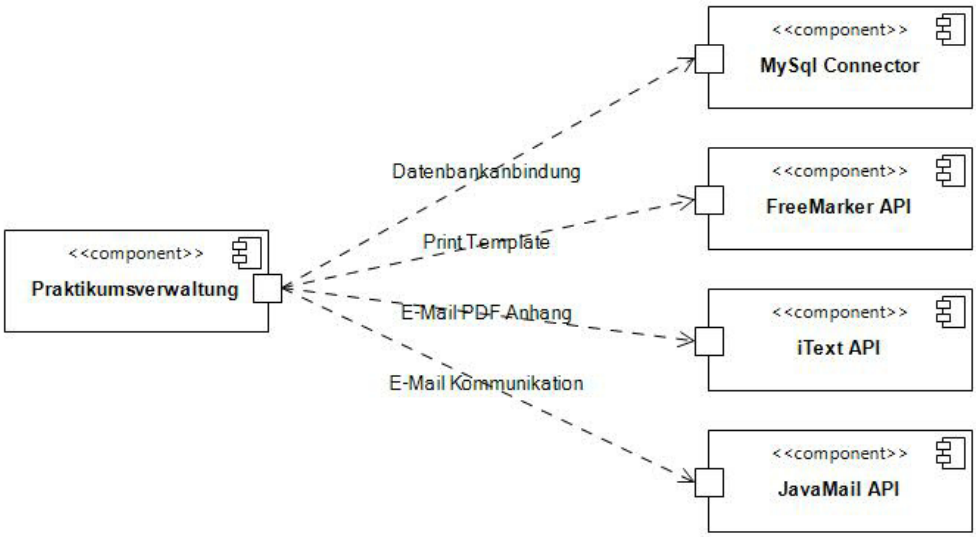


Abbildung 11: Auflistung der verwendeten APIs

4.5 Entwurf der Benutzeroberfläche

Die Benutzeroberfläche besteht aus einem JDesktopPane (Klasse *Praktikumsverwaltung*), welche neue JInternalFrames aufnehmen kann. Die JInternalFrames werden von der Klasse *View* implementiert. Die Views werden über die Klasse *Praktikumsverwaltung* zum JDesktopPane hinzugefügt.

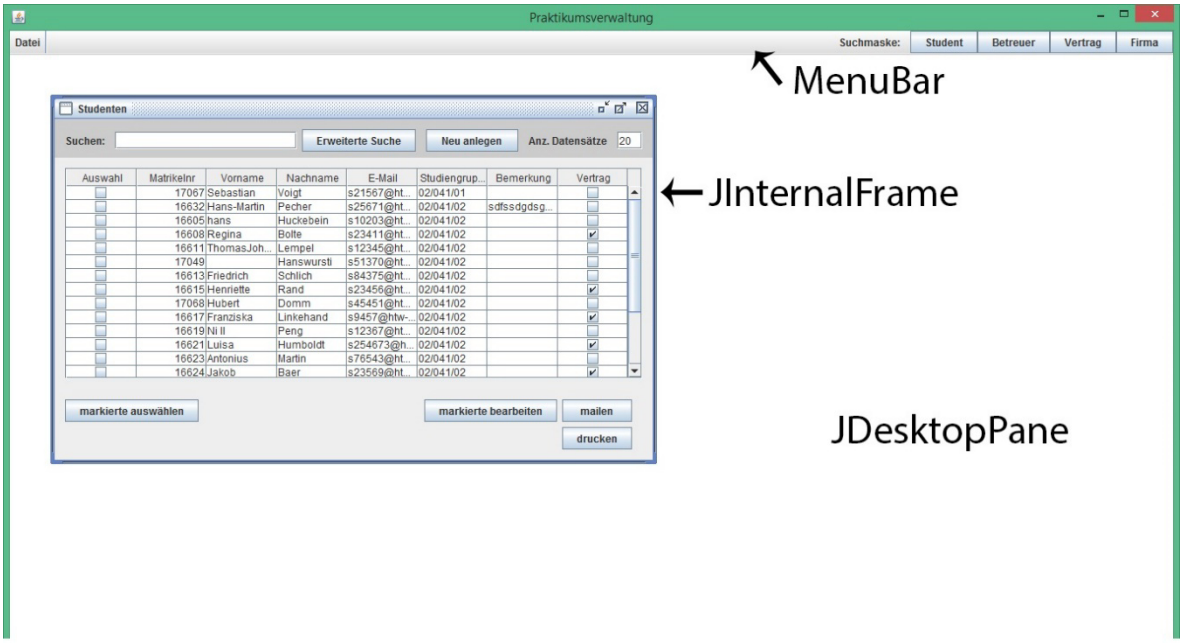


Abbildung 12: javax.swing Hauptkomponenten der GUI

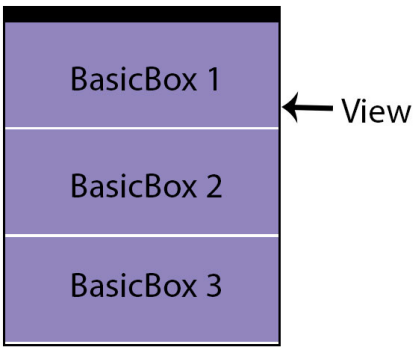


Abbildung 13: Listenstruktur der View

Die Struktur der View basiert auf einem Listenkonzept mit Box-Elementen. Die Liste ist vertikal ausgerichtet und nimmt nacheinander Boxen (Implementation des *BasicBox* Interfaces) auf. Die Boxen verfügen über die komplette Breite der Liste. Es kann also eine beliebige Konstellation von Boxen kombiniert werden.

Möglich ist auch die Schachtelung von Boxen. Die Delegation von Events der Box, obliegt ihr selbst. Es ist möglich die Events an den Controller weiterzuleiten oder selbst zu verarbeiten (in Abhängigkeit der Funktion der Box).

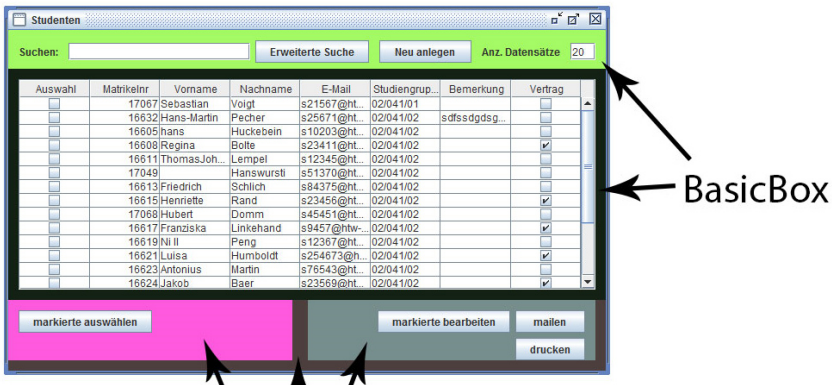


Abbildung 14:View im Debug Modus, zu sehen ist die Box-Struktur und verschachtelte Boxen

Soll der Controller die Events verarbeiten, empfiehlt es sich ein Interface für die Box zu erstellen (mit Endung „Ctrl“) und dieses durch den Controller implementieren zu lassen.

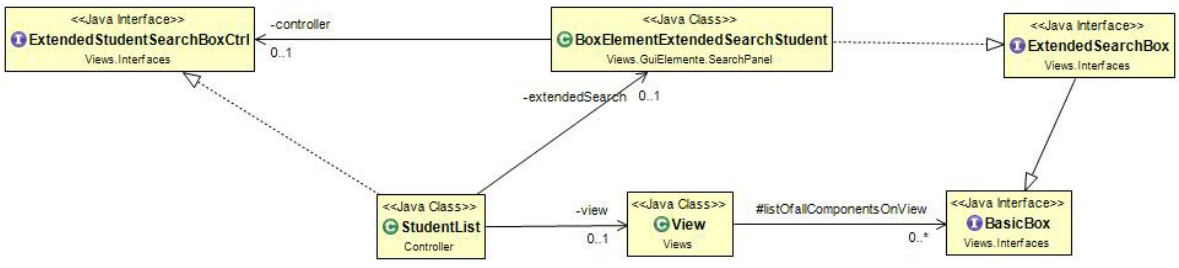
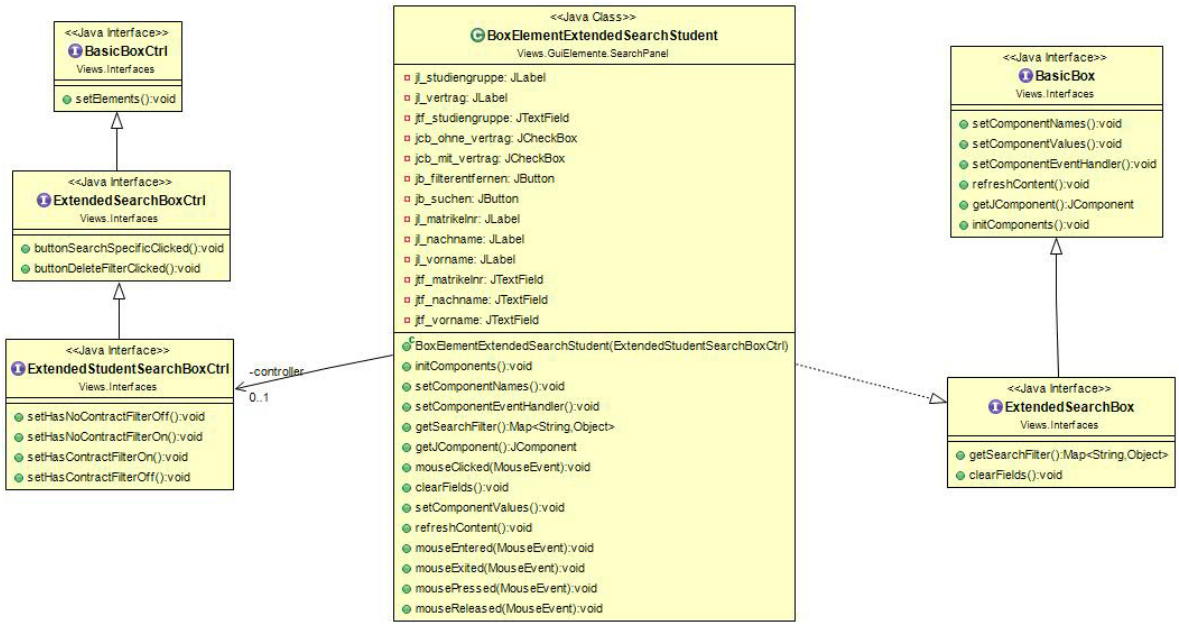


Abbildung 15: Klassenstruktur der Listenansicht des StudentList Controller und der ExtendedSearchBox

Im angeführten Beispiel der Controllerklasse *StudentList* und dem eingesetzten View Element *ExtendedSearchPanelStudent* kann man die Abhängigkeiten gut erkennen. Diese Struktur zieht sich durch alle Box-Elemente.



Die Klassenstruktur der Box *ExtendedSearchPanelStudent* fordert einen *ExtendedStudentSearchBoxCtrl*, welcher über alle Events der Box informiert wird. Hier als Beispiel die Implementation des *mouseClicked* Event der Box.

```
public void mouseClicked(MouseEvent e) {
            if(e.getComponent() == jb_filterentfernen)
                  controller.buttonDeleteFilterClicked();

            if(e.getComponent() == jb_suchen)
                  controller.buttonSearchSpecificClicked();

}
```

Abbildung 16: mouseClicked Event in ExtendedSearchPanelStudent

Die Box ruft hier die im *ExtendedSearchBoxCtrl* definierten Funktionen auf dem Controller aus, sie ist also von der konkreten Funktion entkoppelt.

4.6 Entwurf der Funktionalität/Interaktionsmodell

Die Architektur wurde unter Berücksichtigung des MVC-Patterns entworfen. Daten, Ansichten und Logik wurden so klar wie möglich voneinander getrennt.

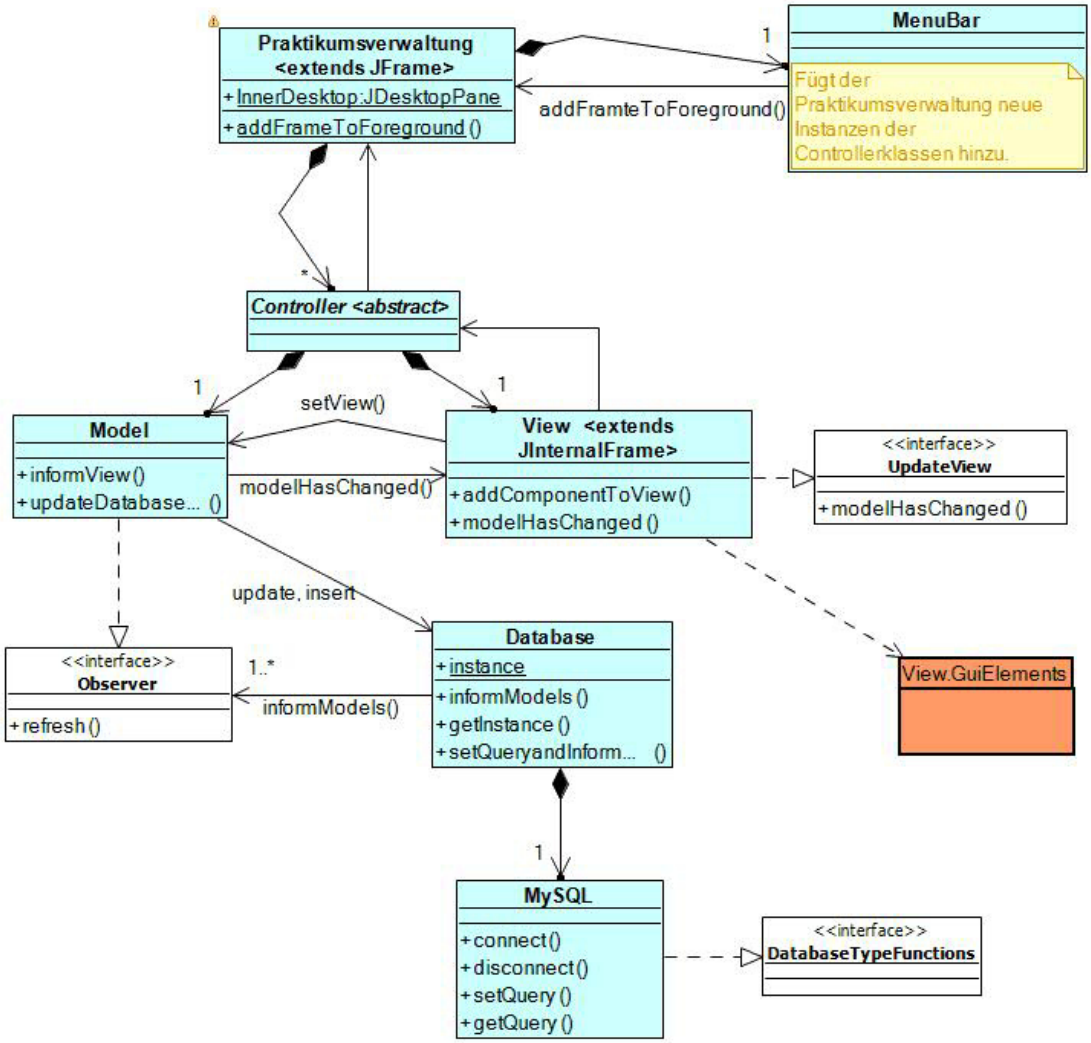


Abbildung 17: Architektur der Software mit den wichtigsten Funktionen und Klassen.

Bsp: Nutzer will vom Startbildschirm nach einem Studenten „Müller“ suchen:

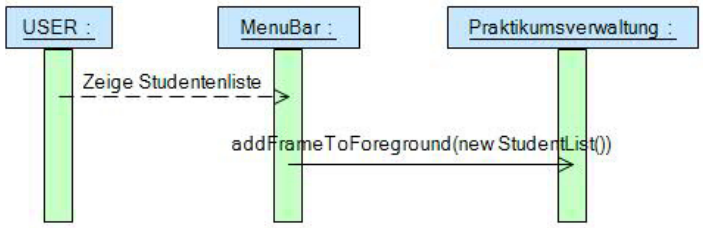


Abbildung 18: User ruft die Listenansicht der Studenten auf.

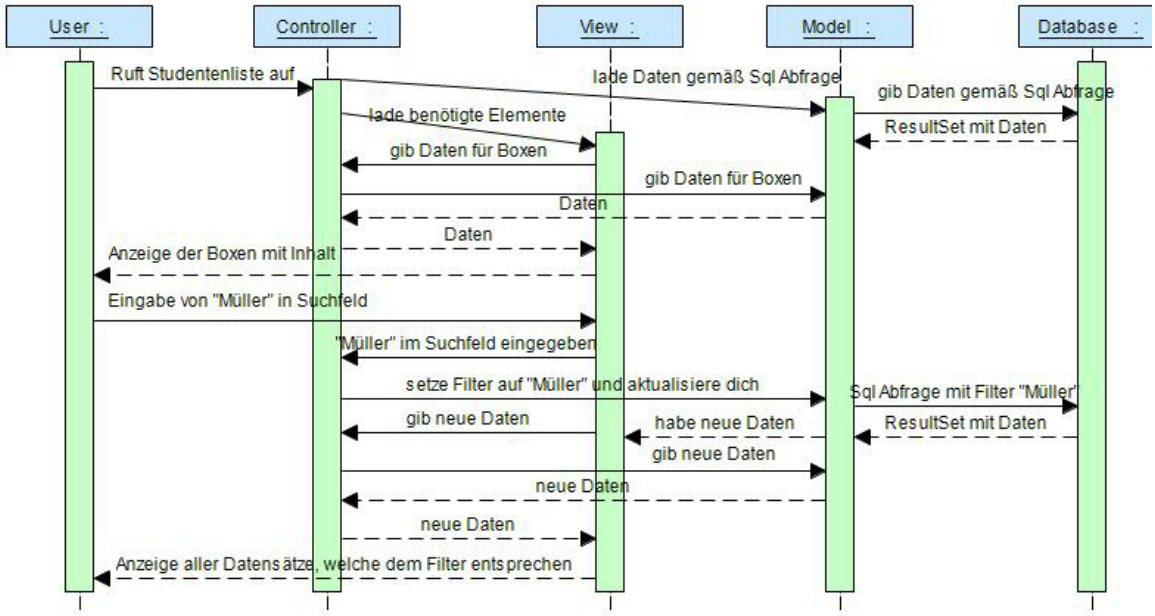


Abbildung 19: Sequenz der internen Aufrufe und Datenflüsse bei der Suche nach „Müller“ in der Listenansicht der Studenten.

4.7 Datenverwaltung / Datenbankentwurf

Die Datenbankstruktur wurde vorgegeben und durfte nicht verändert werden. Ein objektrelationales Mapping fand im Projekt keine Anwendung. Da wir die Daten im Tabellenkontext weiter verwendet haben.

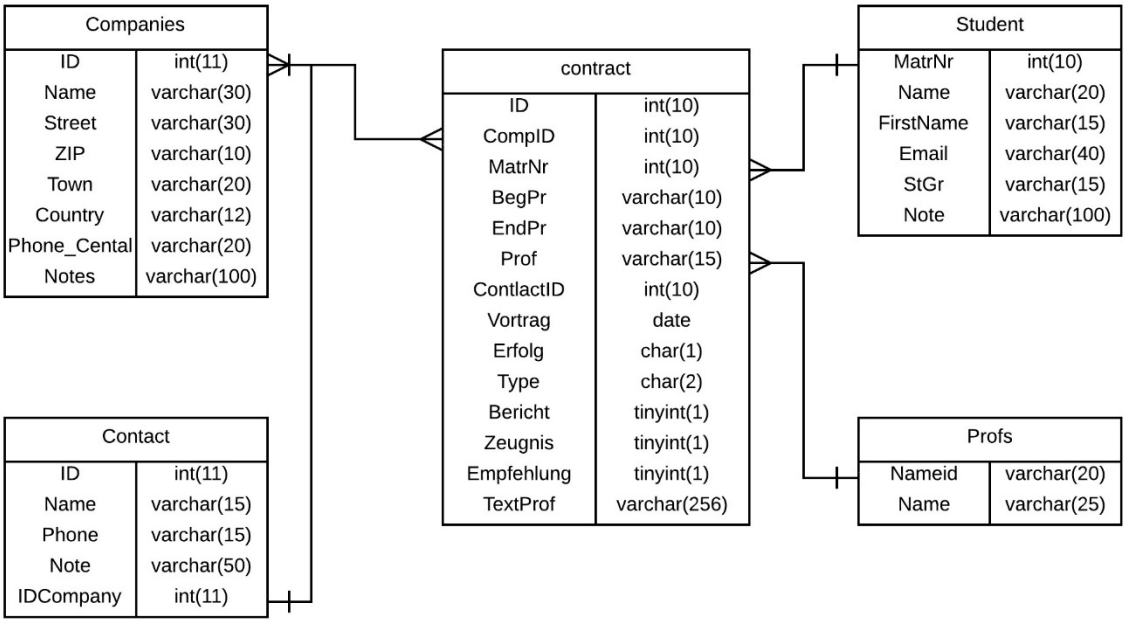


Abbildung 20: Entity-Relationship-Modell der Datenbank

5 Implementation

Es wurde das SMPT – Protokoll für den Email Versand genutzt. Es wuden keine eigenen Protokolle definiert.

5.1 Der Build-Prozess

Um den Quelltext zu kompilieren wurde ein Ant-Script erstellt, welche über die Kommandozeile oder Eclipse gestartet werden kann. Das Script *build.xml* ist im Ordner „scripts“ zu finden.

Voraussetzung für die Ausführung über die Kommandozeile ist das Programm „Ant“, welches installiert[1] sein muss.

Eclipse Kepler bringt ein installiertes Ant bereits mit.

Ausführung des Script über die Kommandozeile:

```
ant –buildfile build.xml
```

Ausführung des Scripts über Eclipse:

Window ->Show View -> Ant -> Add Buildfiles -> build.xml suchen -> Start

Analog zu diesem Prozess kann auch die JDoc neu generiert werden. Dafür kann das Script *javadoc.xml* genutzt werden.

[1] Ant Download über http://ant.apache.org/bindownload.cgi

5.2 API

An dieser Stelle verweisen wir auf die generierte JDoc.

5.3 Teststrategien und -werkzeuge

Zum Testen der Komponenten wurde JUnit in Verbindung mit Mockito eingesetzt. Mockito bietet Möglichkeiten Mock-Objekte komfortabel zu erstellen und auch schwierige Objekte herzustellen (z.B. Instanz der Klasse *ResultSet*).

Tests wurden nur für sehr wenige Funktionen geschrieben. Zu finden sind diese im *src* Ordner unter *test*.

Für die Tests wurde eine extra Ordnerstruktur erstellt, welche mit der Packagestruktur des *src* Ordners identisch ist. Mit dieser Vorgehensweise hat man die Möglichkeit auf Funktionen und Variablen, welche in der *src*-Struktur mit dem Schlüsselwort *protected* geschützt sind zuzugreifen. Voraussetzung dafür ist das hinzufügen der *test* Ordnerstruktur zum Buildpath des Projekts (in Eclipse).

5.4 Testfallspezifikation

Test wurden für folgende Klassen geschrieben:

- ConfigParser.Config
- Controller.Controller
- Controller.Mailing
- Controller.Print
- Models.Datenbank.Database
- Models.Table.TableData

5.5 Testdurchführung und Testergebnisse

Die Tests können in Eclipse gestartet werden.

Zu diesem Zeitpunkt verliefen alle Tests erfolgreich.

6 Anwenderdokumentation

An dieser Stelle verweisen wir auf die eigenständige Anwenderdokumentation.

7 Projektbewertung aus Entwicklersicht

Das Projekt wurde gemäß Pflichtenheft umgesetzt. Verbesserungen können noch im Bereich der internen Strukturierung und Flexibilität erreicht werden.
Hier einige Vorschläge:

- komplette Abstraktionsebene der Datenbank durch Einsatz von Sql-Views
- View-Boxen erben von einer Abstrakten Klasse alle wichtigen Methoden und übernehmen die Swing-Containerklassen (JPanel, etc.) in einer „hat-eine“ Beziehung
- Möglichkeit Mail Templates zu verwalten und auch Mails an Studenten zu und Firmen zu versenden