# Praktikumsverwaltung

Projektdokumentation zum Softwareentwicklungsprojekt

(Entwicklerdokumentation)

Lehrveranstaltung „Software Engineering I / II“

23. Juni 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

**Inhaltsverzeichnis**

1. **Einleitung**
2. **Projektmanagement**
   1. Vorgegebener Zeitablauf
   2. Ressourcenplanung und Organisation
   3. Werkzeugunterstützung
      1. Managementwerkzeuge
      2. Softwareentwicklungswerkzeuge
3. **Pflichtenheft**
4. **Anforderungsanalyse und Entwurf**
   1. Anwendungsfallanalyse
   2. Problembereichsanalyse
   3. Stand der Wissenschaft und Technik
   4. Entwurf der Systemarchitektur
   5. Entwurf der Benutzeroberfläche
   6. Entwurf der Funktionalität/Interaktionsmodell
   7. Datenverwaltung / Datenbankentwurf
5. **Implementation**
   1. Der Build-Prozess
   2. API
   3. Teststrategien und -werkzeuge
   4. Testfallspezifikation
   5. Testdurchführung und Testergebnisse
6. **Anwenderdokumentation**
7. **Projektbewertung aus Entwicklersicht**

**Literatur**

Einleitung

Beschreibung des Vorhabens. Erlauterungen zur Entwicklung und Nutzung des Projekts. Ein-¨ ordnung der geschaffenen Losung in den wissenschaftlichen Kontext.¨

# 2 Projektmanagement

## 2.1 Vorgegebener Zeitablauf

|  |  |
| --- | --- |
| Vorlage Pflichtenheft | 31.01.2014 |
| Bearbeitung Analyse/Entwurf: |  |
| Klassendiagramme, Komponentendiagramme,  Paketstruktur; Verteilungsdiagramme, ... Festlegung der  Rollenverteilung |  |
| Vorlage Projektdokumentation (Zwischenstand) | 06.05.2013 |
| Durchführung der Implementierung  User Interface Prototyping Bereitstellung API (Javadoc) |  |
|  |  |
| Gruppenkolloquium | 06.05. + 17.05.2013 |
| Implementation, Testfallspezifikation, Testung (JUnit) |  |
|  |  |
| Abgabe der Projektdokumentation | 23.06.2014 |
| Vorbereitung Präsentation¨ |  |
|  |  |
| Präsentation | 26.06.2013 |

## 2.2 Ressourcenplanung und Organisation

Einsatz personeller Ressourcen, Rollenverteilung im Projekt, Zuordnung von Aufgaben zu Rollen, inhaltliche Schnittstellen.

# sepp: import und gui  
# rick: datenbank und modularität  
# ivo: druck  
# jakob: mail

Das Projekt wurde von vier Studenten in einem Studiensemester angefertigt. Wichtige Projektentscheidungen wurden im Kollektiv getroffen.

Folgende Rollen wurden verteilt:

|  |  |  |
| --- | --- | --- |
| Rolle | Mitarbeiter | Aufgaben |
| Projektleiter |  |  |
| Usability Manager |  |  |
| Architekt |  |  |
| Programmierer |  |  |
| Tester |  |  |
| Qualitätsmanagement |  |  |
|  |  |  |

## 2.3 Werkzeugunterstützung

### 2.3.1 Managementwerkzeuge

Als Managementwerkzeuge wurden eingesetzt:  
- Git 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**

#Hier wird auf das als gesondertes Dokument vorgelegte Pflichtenheft verwiesen.

# 4 Anforderungsanalyse und Entwurf

## 4.1 Anwendungsfallanalyse

Detailliertere Beschreibungen der UseCases des Pflichtenheftes,   
Darstellung durch Aktivitats-¨ diagramme.   
# sind soweit vorbereitet  
  
Beschreibung von Anwendungsszenarien,   
Darstellung durch Sequenzdiagramme.  
#bsp einer bearbeitung, neuanlage, import?  
  
Insbesondere ist darzustellen, wer die Akteure sind und welche Aufgaben sie jeweils haben.  
# wir haben nur prakt.beauftragten -> macht alles

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.

Alle Bereiche:

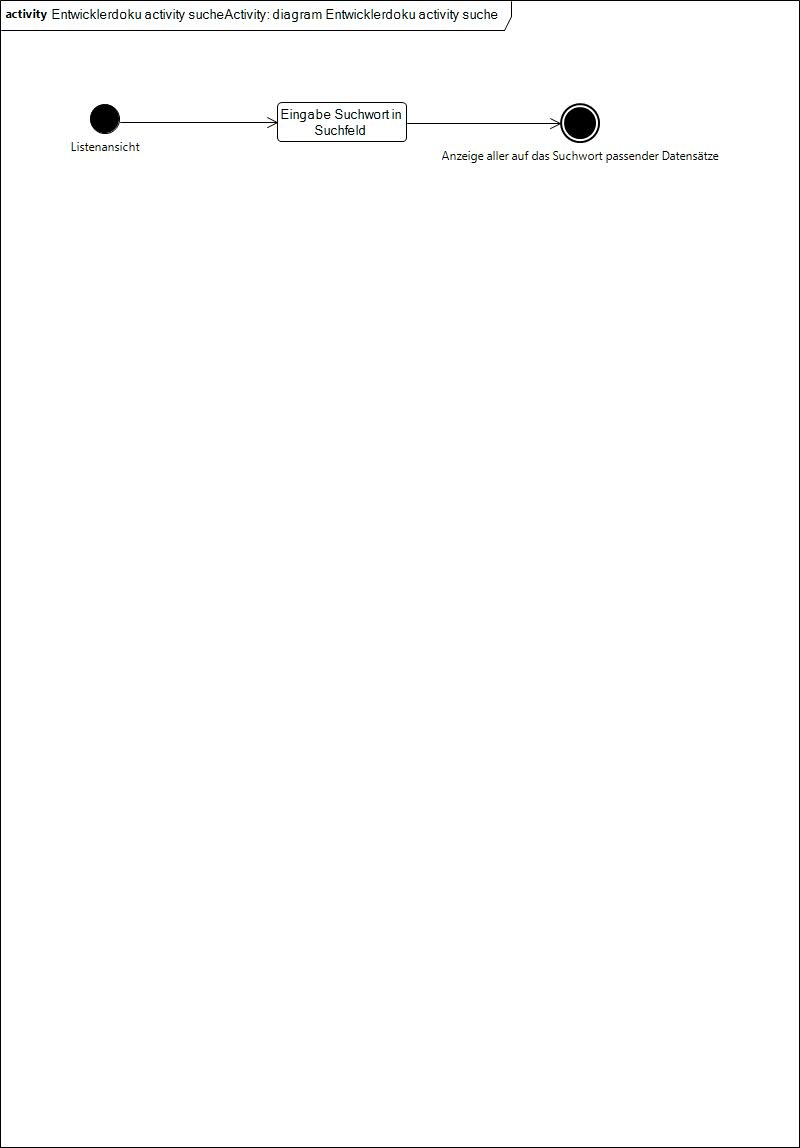


Abbildung : Suche über Suchfeld

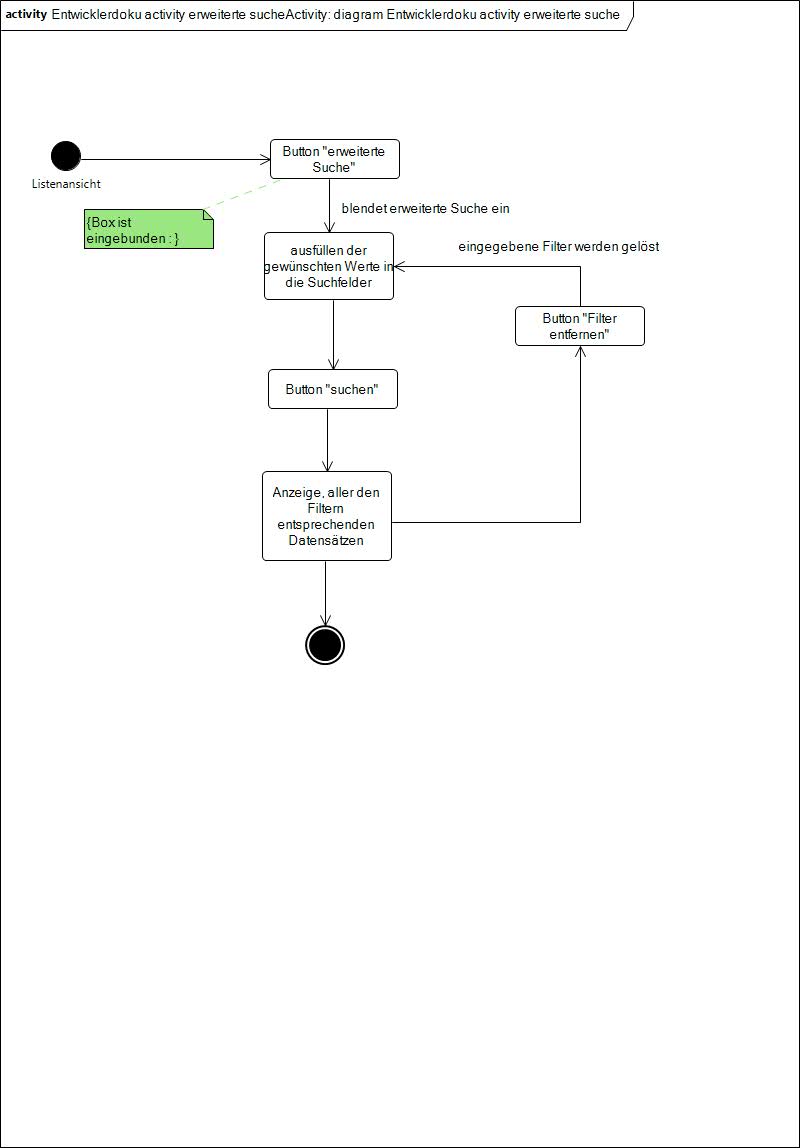


Abbildung : Erweiterte Suche

Bereich Student:

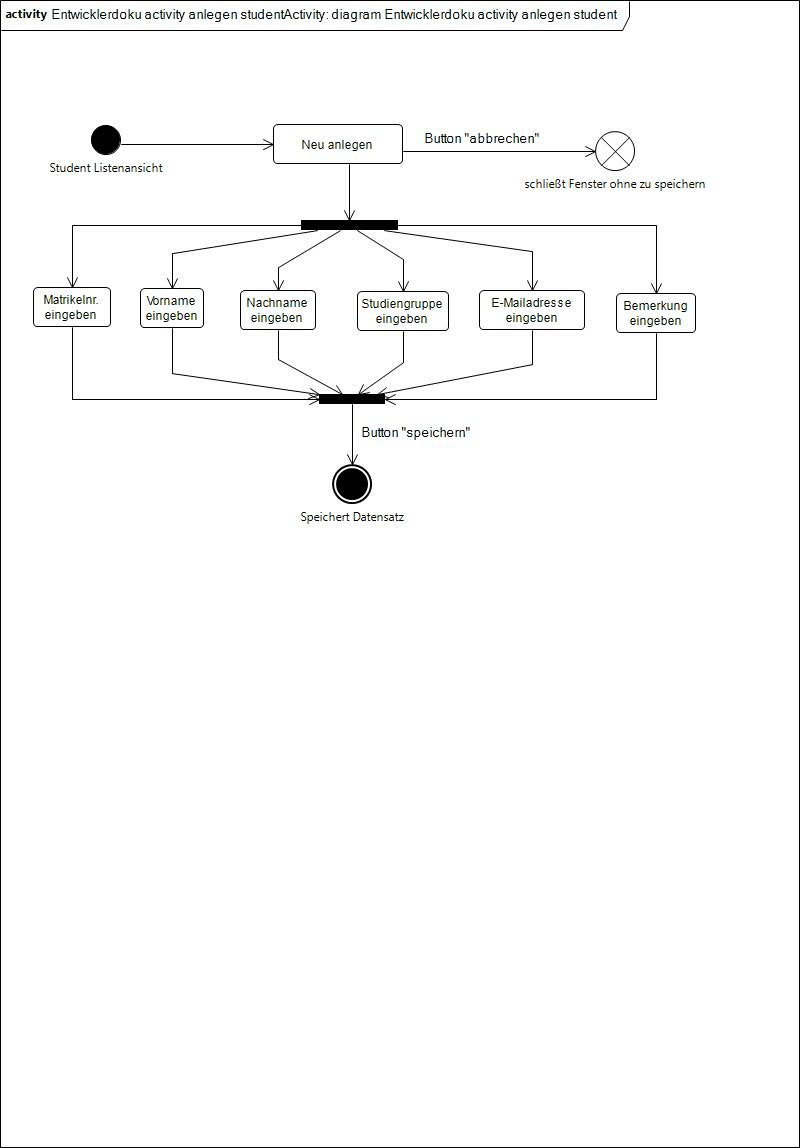


Abbildung : Student anlegen

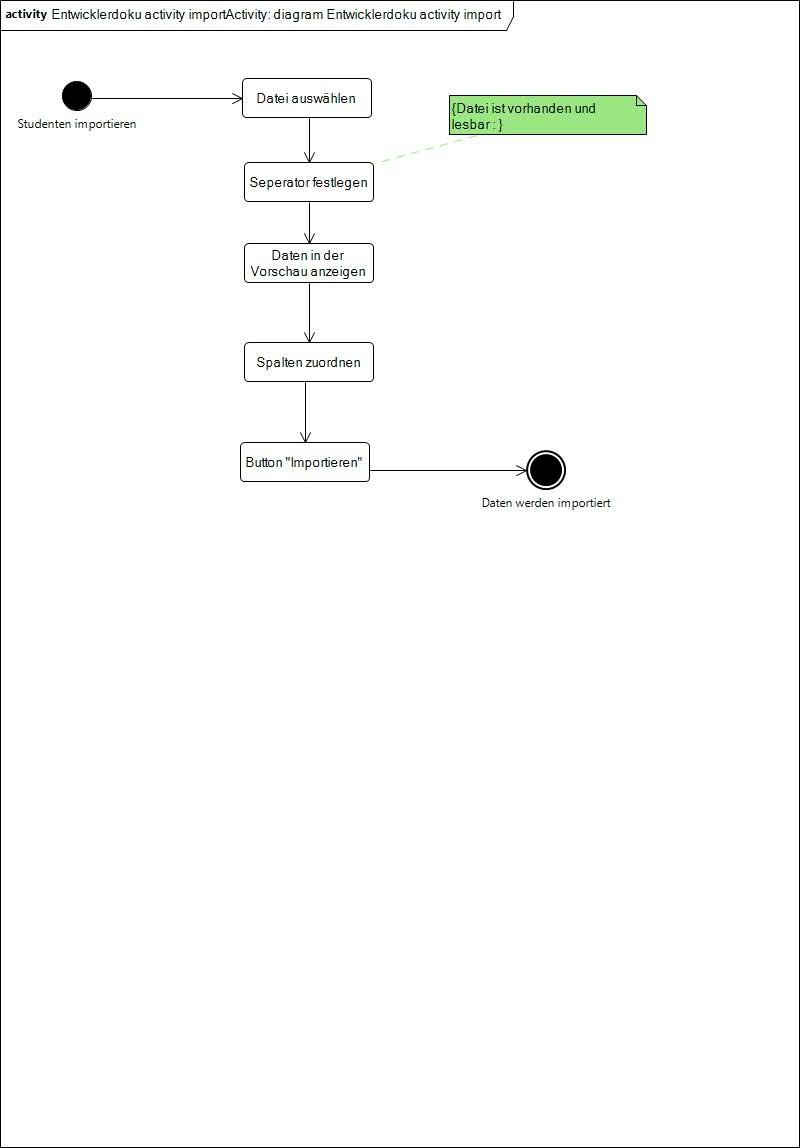


Abbildung : Import von Studentendaten aus einer Datei.

Bereich Betreuer:

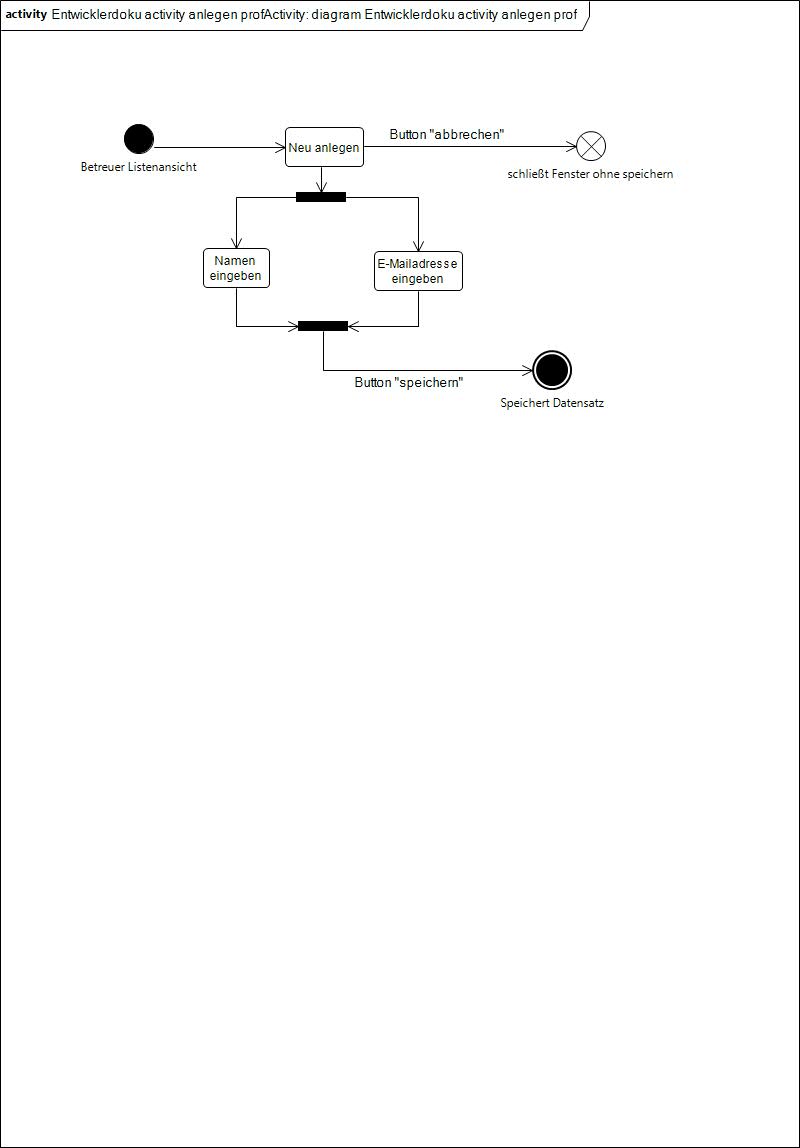


Abbildung : Betreuer anlegen

Bereich Firma:

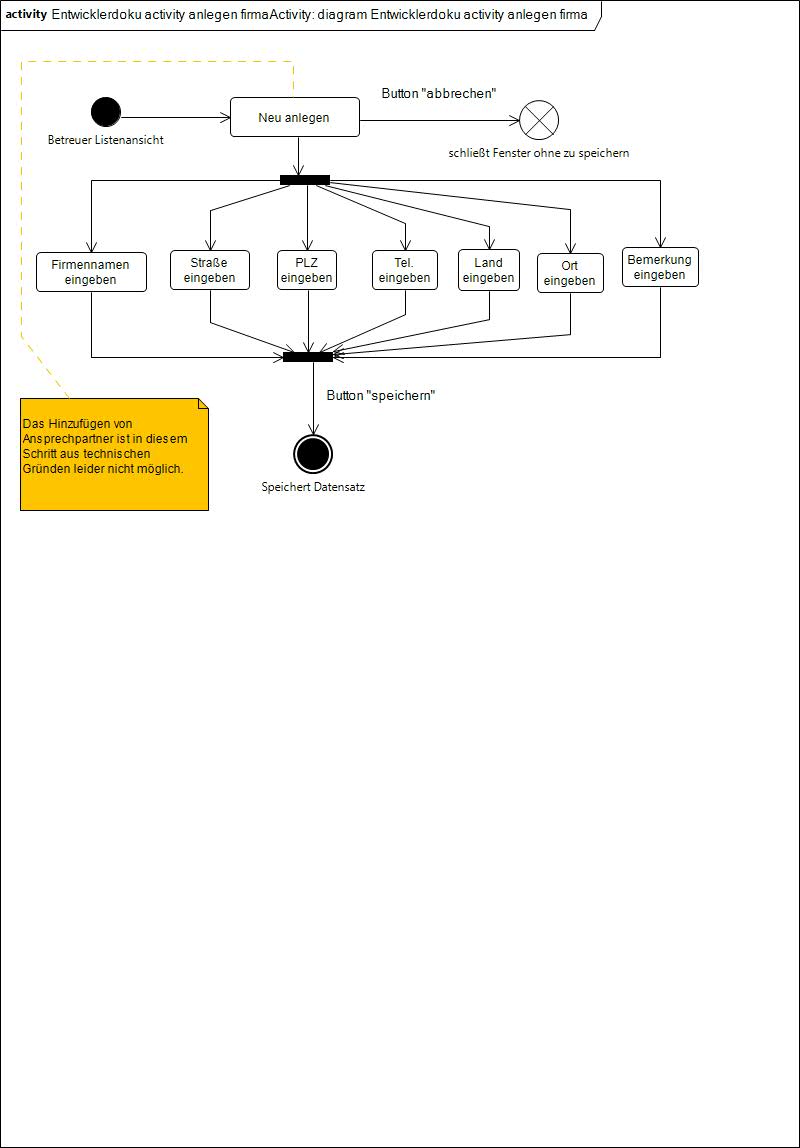


Abbildung : Firma anlegen

Bereich Verträge:

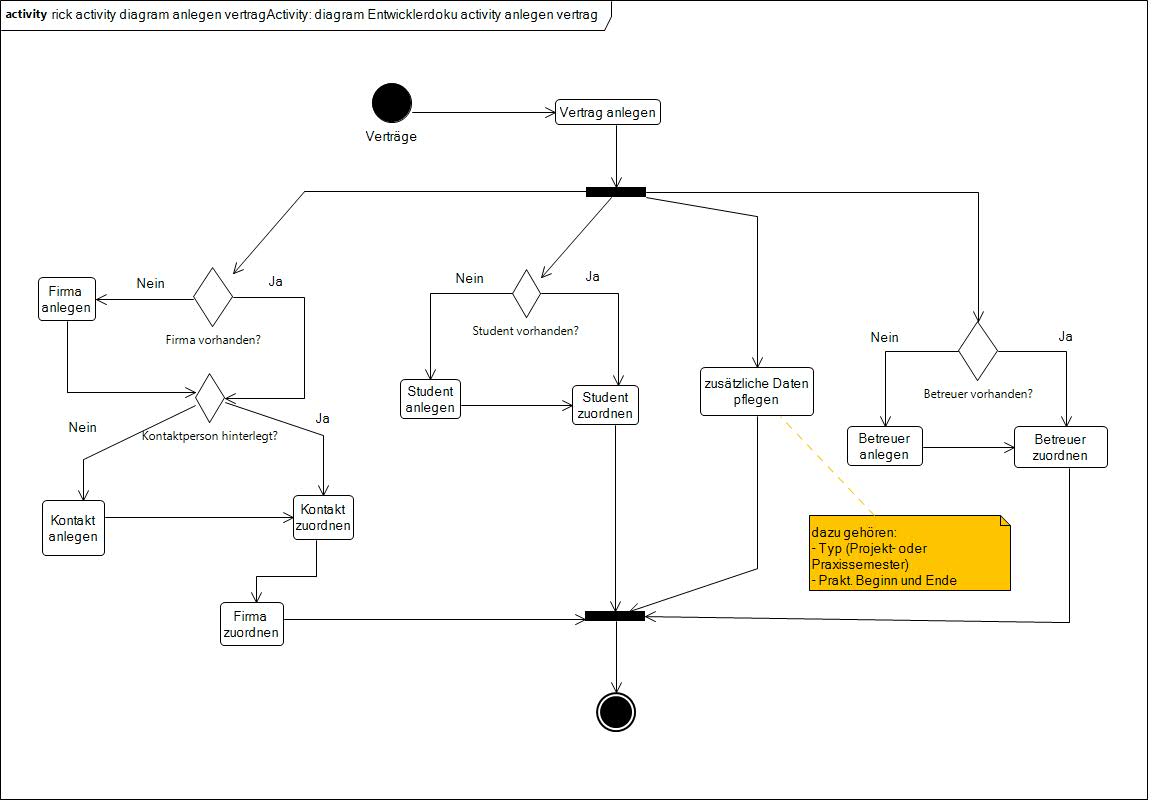


Abbildung : Vertrag anlegen

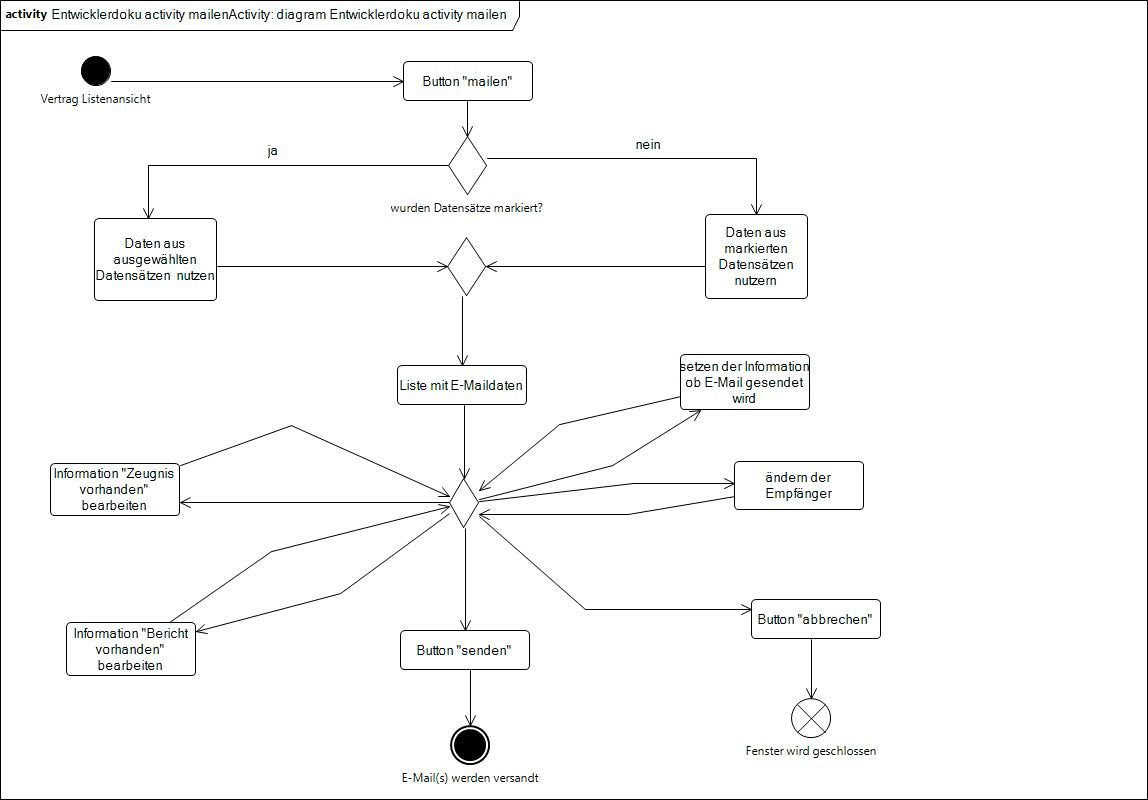


Abbildung : E-Mails versenden

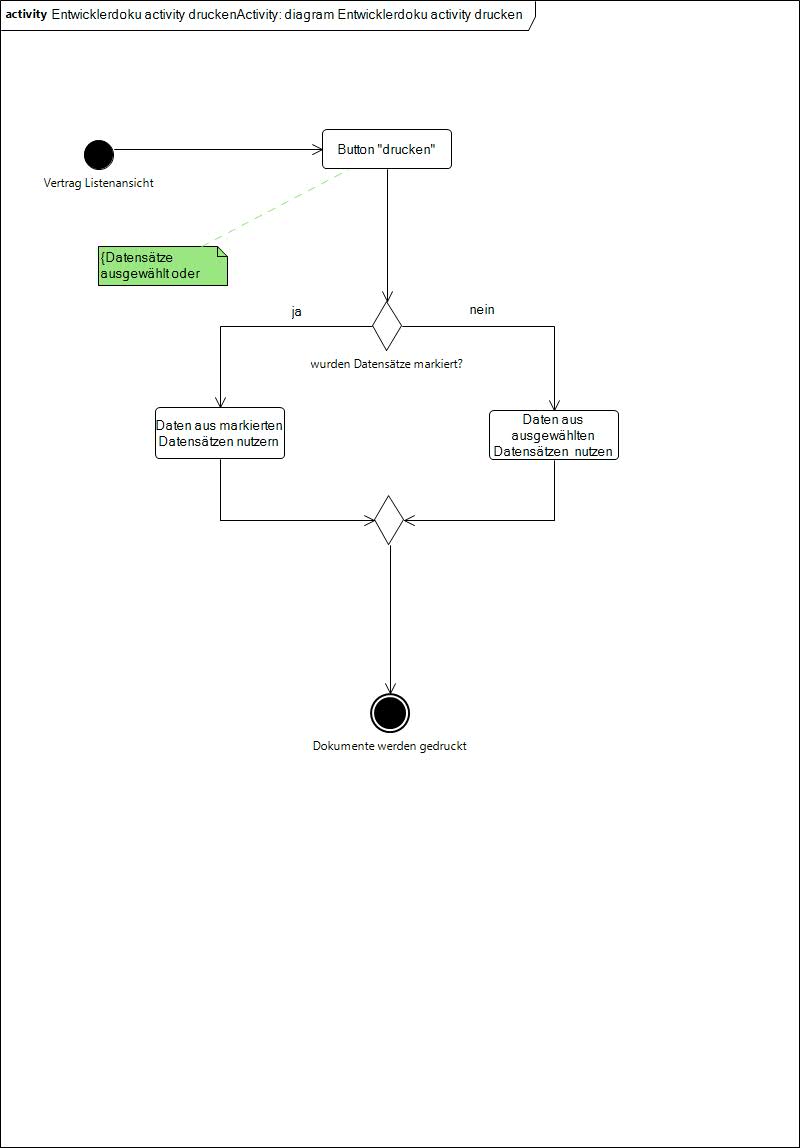


Abbildung : Dokument drucken

Beschreibung von Anwendungsszenarien:

Folgende Szenarien werden beleuchtet:  
- manuelle Neuanlage von Studenten

4.2 Problembereichsanalyse

Analyse und Strukturierung des Problems. Identifizierung von Teilproblemen. Identifizierung von Klassen. Hier sind besonders verständliche verbale Beschreibungen notwendig!¨  
# herleitung der überlegung zu listen und gleichheit derer  
  
Der Problembereich des Projektes „Praktikumsverwaltung“ umfasst den Informationsfluss und die Informationshaltung von Verträgen zu Praktika- und Projektsemestern von Studenten.  
  
Ein Vertrag besteht aus Informationen zum Vertrag1 und der Zuordnung einer Firma, eines Studenten und eines Betreuers.   
  
Analog dazu werden die detaillierten Informationen zu Firmen, Studenten und Betreuern ebenfalls in tabellarischer Form in der Datenbank gehalten.  
  
Die Daten liegen bereits in einer relationalen Datenbankstruktur vor und welche unverändert genutzt werden sollte.

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

Überlegungen zur Abwägung zwischen Nachnutzung und Eigenentwicklung. Dieser Punkt kann recht umfangreich sein, wenn vorhandene Losungen erst analysiert werden müssen (z.B. CMS Projekt).  
  
Da wir auf keinen 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

Grobarchitektur, z.B. Darstellung eines Schichtenmodells. Komponentenbildung. Nachnutzung von Bibliotheken/Frameworks. Verteilungsdiagramm(e) im Falle von Client-ServerAnwendungen.  
#componentdiagram

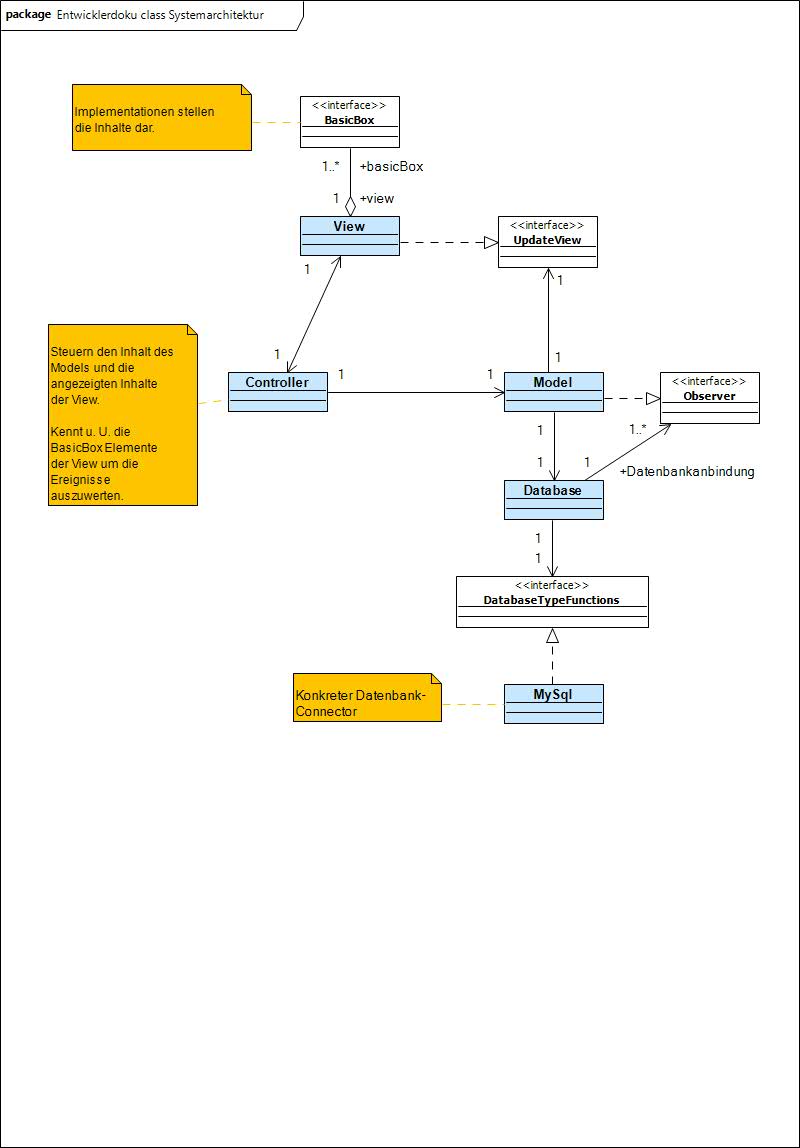
Um eine modulare Softwarearchitektur zu erhalten und zudem eine Trennung von Funktion und Design zu gewährleisten, wurde das Architekturmuster „Model-View-Controller“1 angewandt.  
  
In unserer Umsetzung übernimmt der Controller die Zusammensetzung der Anzeige, die Reaktion auf Ereignisse dieser und 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 „Singleton2“-Architekturmuster implementiert und nutzt zusätzlich die Eigenschaften des „Observer3“-Musters um alle Models (welche das Interface Observer implementieren) bei einer Änderung der Daten zu informieren.  
  
  


Abbildung : 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.

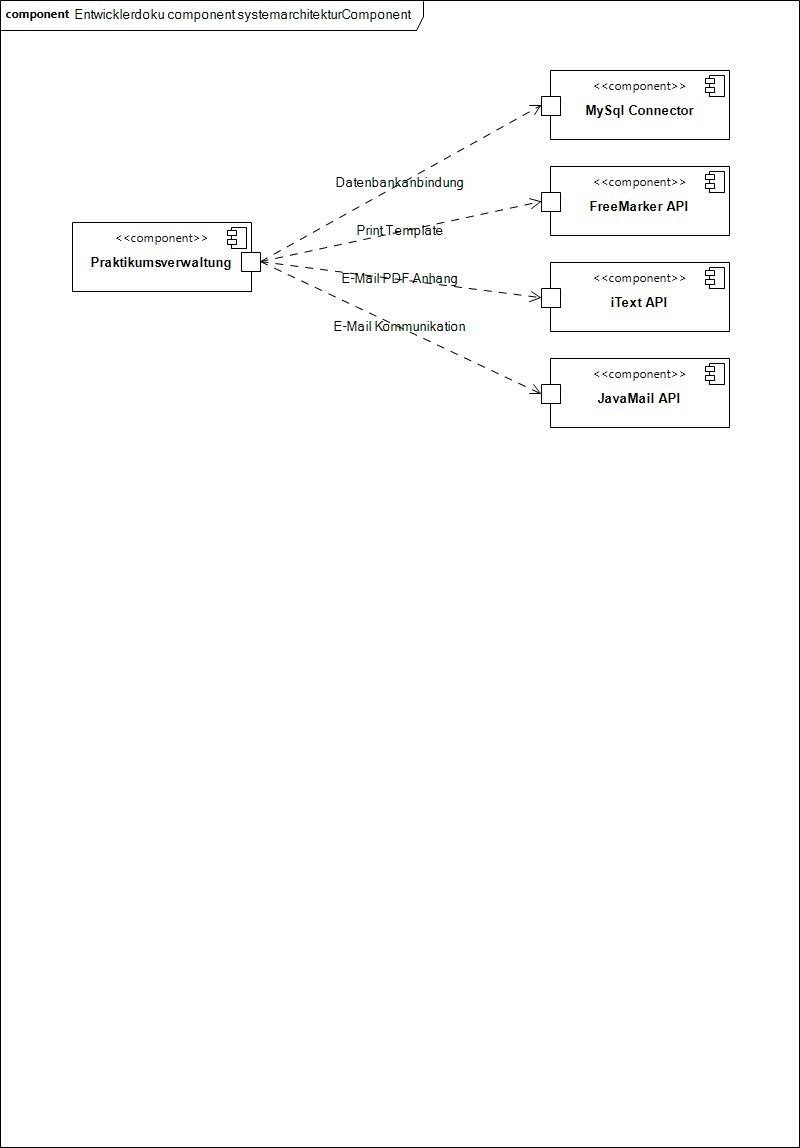


Abbildung : Auflistung der verwendeten APIs

## 4.5 Entwurf der Benutzeroberfläche

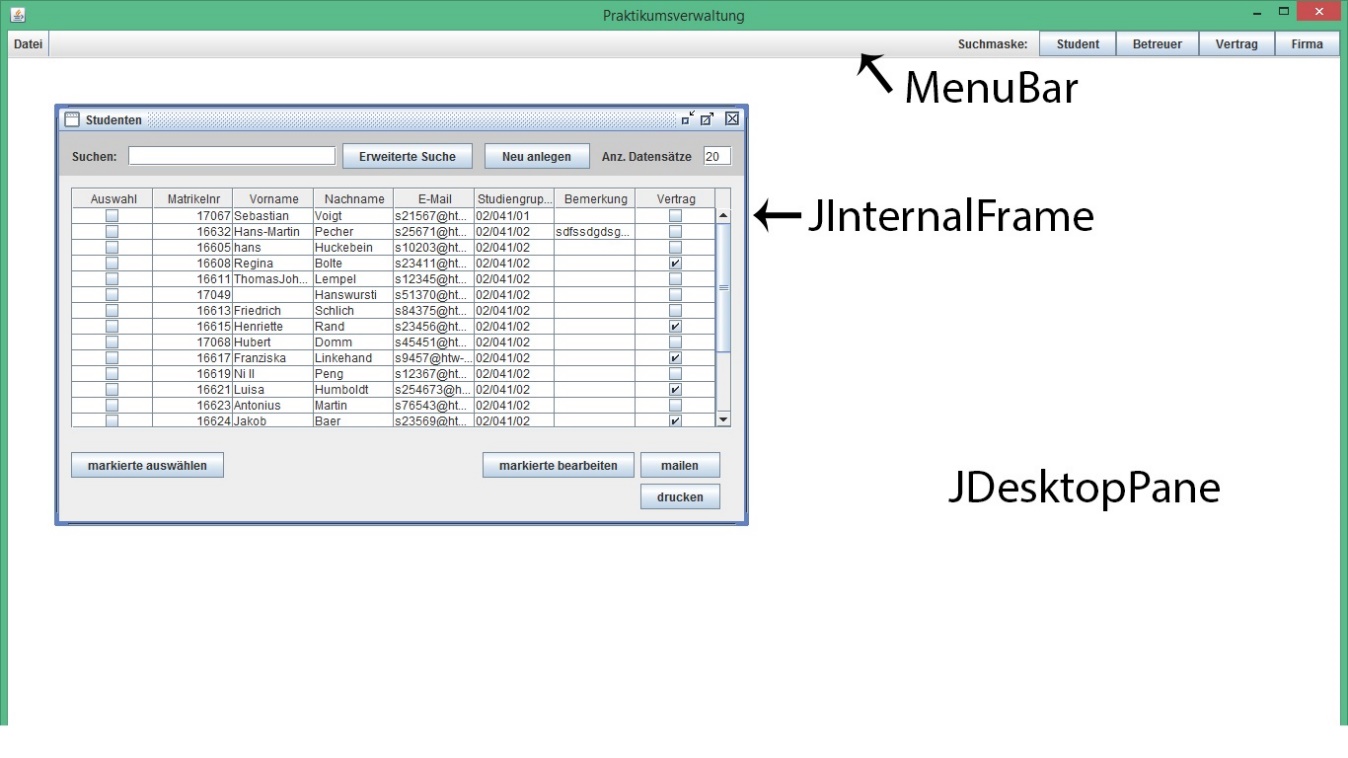
Grafische Darstellung der Struktur der GUI und der über die GUI ausgelösten Aktivitäten. Hier sind bereits die Klassen für die Komposition der Oberflächenelemente anzugeben, auch die Struktur von Menus usw. Darstellung der Dialoggestaltung.  
  
Die Benutzeroberfläche besteht aus einem JDesktopPane (Klasse *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

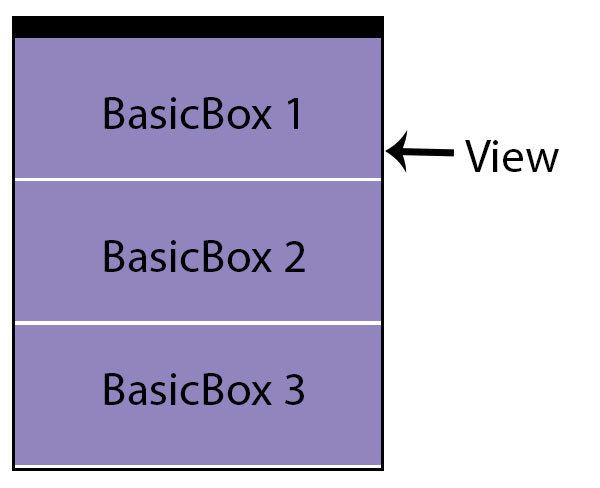
­Die Struktur der View basiert auf einem Listenkonzept mit Box-Elementen. Die Liste ist vertikal ausgerichtet und nimmt nacheinander Boxen (Implementation des *BasicBox* Interfaces) auf. Die Boxen verfügen über die komplette Breite der Liste. Es kann also eine beliebige Konstellation von Boxen kombiniert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

Abbildung 13: Listenstruktur der View

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).

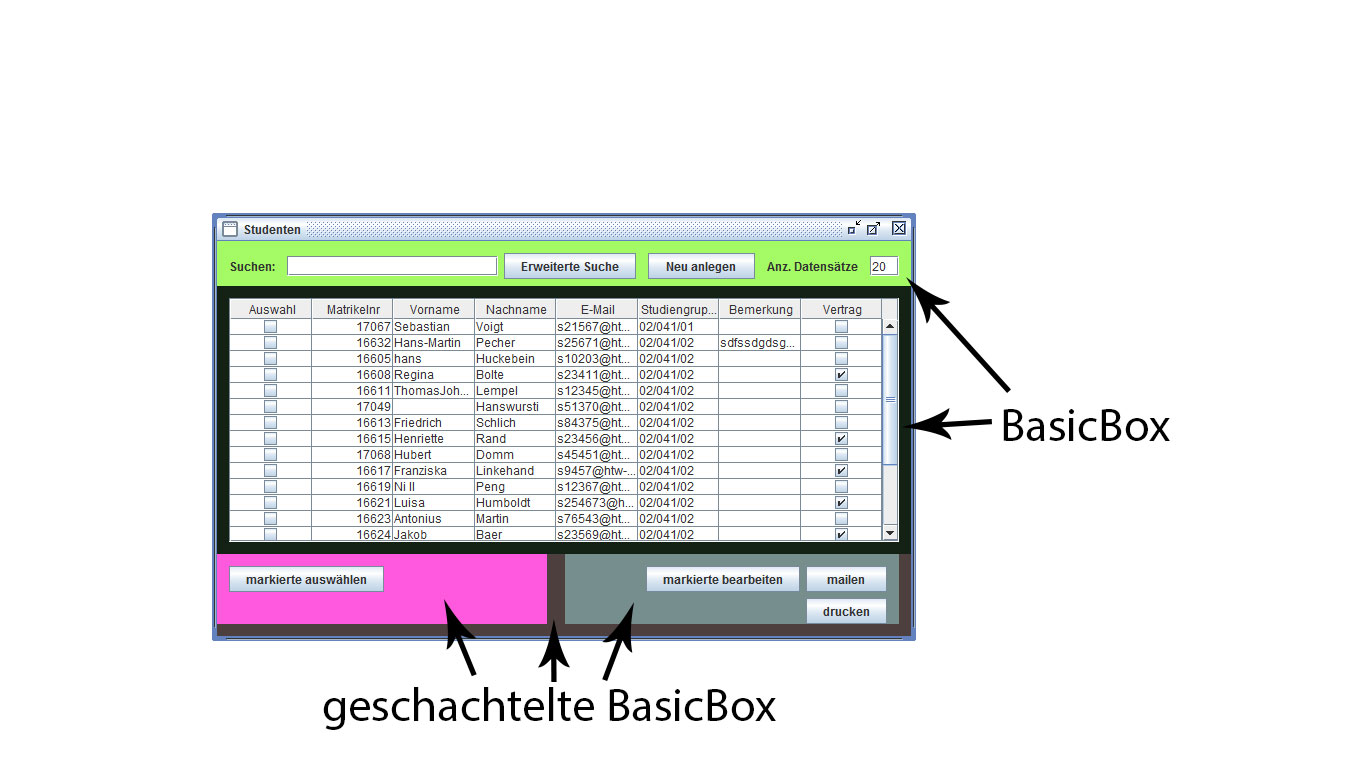
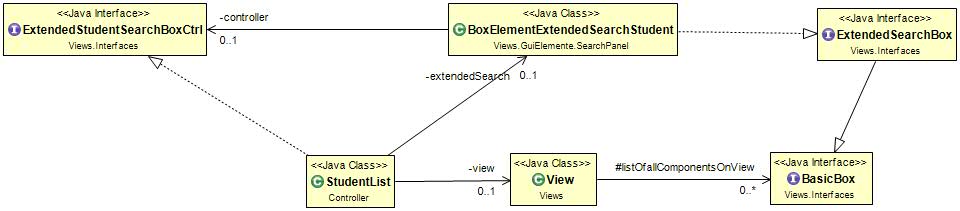
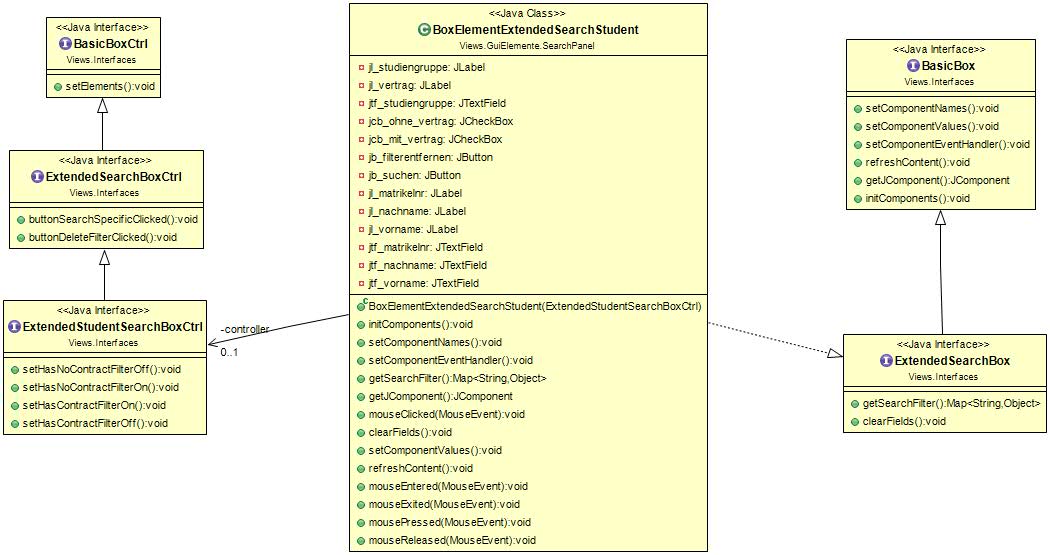
  
  
  
  
  
  
  
  
  
  
Soll der Controller die Events verarbeiten, empfiehlt es sich ein Interface für die Box zu erstellen (mit Endung „Ctrl“) und dieses durch den Controller implementieren zu lassen.   
  


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

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.

Abbildung 16: mouseClicked Event in ExtendedSearchPanelStudent

**public** **void** mouseClicked(MouseEvent e) {

**if**(e.getComponent() == jb\_filterentfernen)

controller.buttonDeleteFilterClicked();

**if**(e.getComponent() == jb\_suchen)

controller.buttonSearchSpecificClicked();

}

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

Beschreibung der logischen Struktur der Anwendung, Klassendiagramme, Zustandsdiagramme, Schnittstellenspezifikationen (Interfaces!) und verwendete bzw. selbst definierte Protokolle.

## 4.7 Datenverwaltung / Datenbankentwurf

Beschreibung der persistenten Datenspeicherung. Angabe von ER-Diagrammen und des ORMappings.

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.

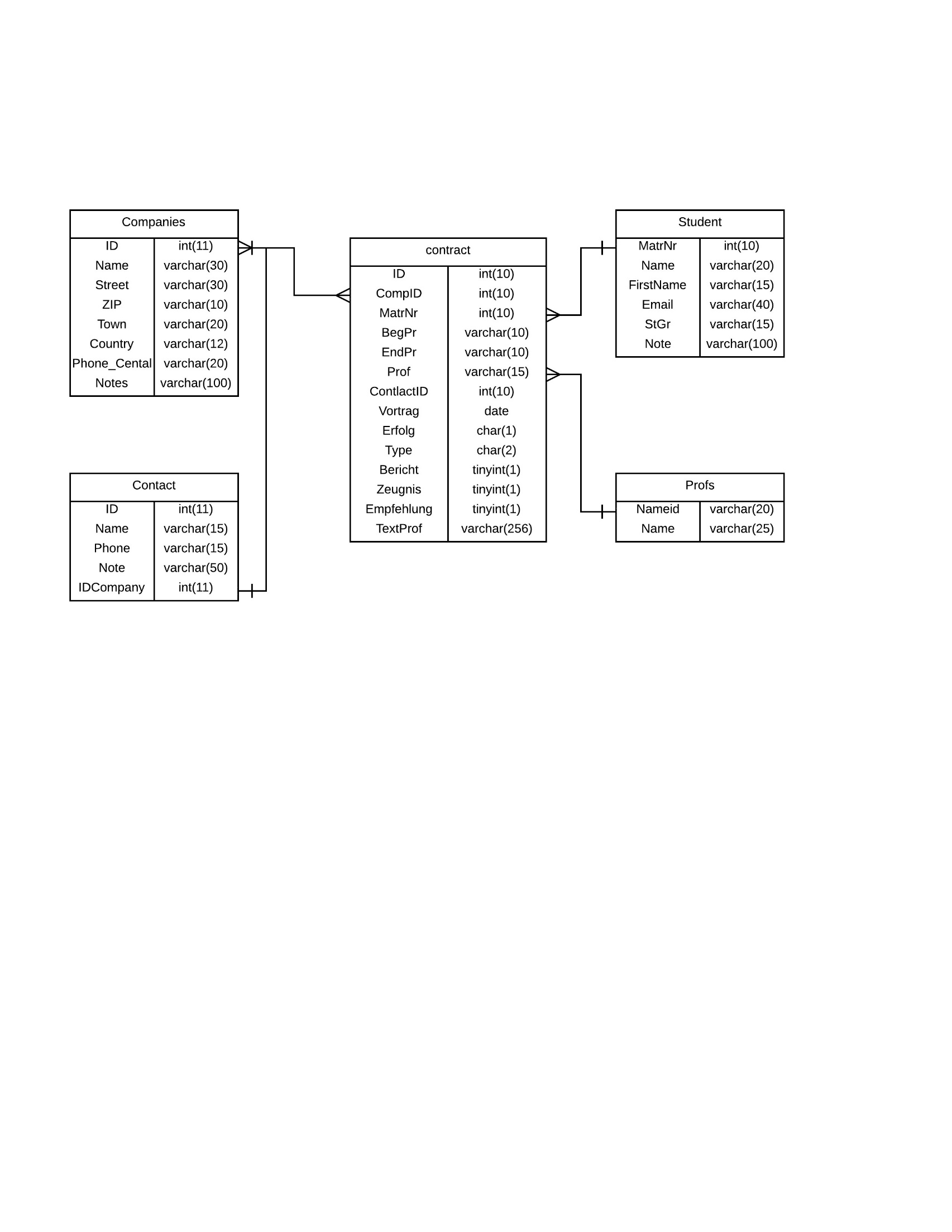


Abbildung : Entity-Relationship-Modell der Datenbank

# 5 Implementation

Abgrenzung von Geschäftsklassen und Fachklassen. Erstellung bzw. Generierung der Quellco-¨ des.

Implementierung von Algorithmen.

## 5.1 Der Build-Prozess

Beschreibung der Erstellung der auszuliefernden Anwendung. Enthalt auch eine Installations-¨ anleitung.

#anleitung ant und jdoc erzeugung  
#installationsanleitung können wir aus der anwenderdoku übernehmen

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 installiert1 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**

#verweis auf jdoc, gesamtübersicht uml und howto tutorials

An dieser Stelle verweisen wir auf die generierte JDoc Dokumentation und die Gesamtübersicht der Klassenstruktur.

**5.3 Teststrategien und -werkzeuge**

Hier wird beschrieben, ob und wenn ja welche Testwerkzeuge eingesetzt werden (z.B. JUnit).

#junit, mockito testframework

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

Hier sollte eine Anleitung zur Installation enthalten sein. Empfohlen wird die Erstellung einer Web-Site fur das Projekt.¨

#Hier verweisen wir auf die separate Anwenderdoku von Sepp

An dieser Stelle verweisen wir auf die eigenständige Anwenderdokumentation.

# 7 Projektbewertung aus Entwicklersicht

Einschatzung des Entwicklungsstandes des Projekts. Beantwortung der Frage¨ Was soll als

” nachstes getan werden, wenn noch Zeit bzw. Geld zur Verf¨ ugung gestellt wird?”¨

# Stichpunke:

* Datenbankstruktur überarbeiten (datentypen vereinheitlichen, auto\_increment auf PK setzen, FK richtig einsetzen)
* Komplette abstraktion der datenbank durch einsatz von Sql-Views
* Boxen erben von einer abstract class alle wichtigen methoden und übernehmen den container (jpanel etc.) als hat-eine beziehung -> größerer freiheitsgrad und bessere flexibilität

Verbesserungsvorschlage, z.B. zur Gestaltung der Entwicklerdokumentation.¨

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

# Literatur