Dokumentation

im Studiengang

Informatik Master

Praktikumsdokumentation: Enterprise Software Architecture

Referent : Prof. Dr. Ulf Schreier

Vorgelegt am : 30.06.2017

Vorgelegt von : Illya Strunskyy 256 316 INM,
Konstantin Vinogradov 256 317 INM

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

JPA Java Persistence API

JSF Java Server Faces

CRUD Create Read Update Delete

ORM Object Relational Mapping

UI User Interface

1 Einführung und Ziele

Im Rahmen des Praktikums von „Enterprise Software Architektur“ sollte ein Projekt durchgeführt werden. Das Thema war weitesgehend frei wählbar. Es sollten allerdings gängige Java EE Technologien eingesetzt werden.

Ziel des Projektes ist es ein Framwork zu entwickeln, welches die Erstellung von Anwendungen zur Verwaltung von Datenbeständen erleichtern soll. Folglich soll mit Hilfe des zu entwickelnden Frameworks die Erstellung von Create Read Update Delete (CRUD)-Anwendungen bestimmten Typs umgesetzt werden können. Das Framework benötigt auf der einen Seite ein existierendes Datenmodel, welches ein Anwender zunächst erzeugen muss. Dieses Datenmodell implementiert ein bestimmtes Interface, welches es dem Framework erlaubt wichtige Informationen aus dem Model zu lesen und für die Benutzeroberfläche zur Verfügung zu stellen. Auf der anderen Seite muss ein Anwender zusätzlich Objekt-Vorlagen im *XTHML*-Format erstellen. Dabei benötigt jede Klasse, die im Webinterface sichtbar sein soll, eine solche Vorlage. Das Framework selbst verwendet das Datenmodel und die dazugehörigen Vorlagen um eine Benutzeroberfläche zu generieren, in welcher alle CRUD-Befehle ausgeführt werden können.Der Datenbestand wird mittels Java Persistence API (JPA) gesichert.

Für das Projekt wurde zum einen das Framework selbst entworfen und zum anderen wurde eine Referenz-Implementierung des Datenmodells erstellt, die die Funktionalität des Frameworks unter Beweis stellt.

2 Evaluations-Beispiel - Seminarverwaltung

Um wie bei einer üblichen Applikation zu beginnen, wird zunächst das zu Grunde liegende Datenmodell beschrieben. Für die Evaluation des Frameworks wurde eine Referenz-Anwendung erstellt, die die Funktionalität des Frameworks unter Beweis stellen soll.

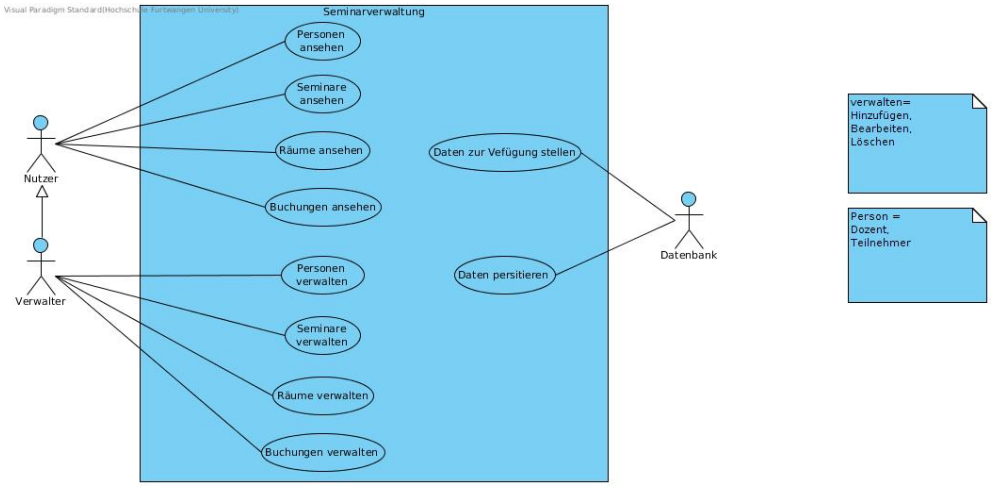
2.1 Beschreibung

Es wurde eine simple CRUD-Anwendung zur Seminarverwaltung erstellt. Grob gesagt, soll es z.B. einem Dozenten möglich sein, Räume für Veranstaltungen zu buchen. Dieses beinhaltet die Erstellung und Verwaltung von den Personeninformationen. Es gibt Teilnehmer, sowie Dozenten die Seminare durchführen. Zusätlich muss das System in der Lage sein, Informationen über die Räume zu bearbeiten, in denen die Veranstaltungen stattfinden. Ein Raum wird über eine Buchung reserviert. Eine Buchung ist dabei die Zuordnung einer Veranstaltung zu einem bestimmten Raum. Die Veransaltung selbst ist an den Dozenten gebunden und beinhaltet zusätzlich die Teilnehmerinformationen. Eine Veranstaltung ist Teil eines Seminars. Ein Seminar stellt dabei ein gesamtes Modul eines Semester dar. Für gewöhnlich enthält ein solches Modul zwei Veranstaltungen, nämlich eine Vorlesung und ein Praktikum.

2.2 Anforderungen

Die Hauptanfoderungen lassen sich aus dem Anwendungsfall Diagramm 1 auslesen. Es gibt drei Akteure des Systems. Einen Benutzer, der vorhandene Daten einsehen kann. So kann dies ein Student sein, der nachsehen will, wann und in welchem Raum er eine Vorlesung hat. Vom Nutzer abgeleitet ist der Verwalter. Dieser ist in der Lage Informationen hinzuzufügen, zu löschen, und zu ändern. Dabei ist es ihm gestattet die Teilnehmer bzw. Dozenten zu verwalten. So wie Seminare, Räume und Buchungen. Damit eine Aufteilung in Nutzer und Verwalter möglich ist, benötigt man einen Login Mechanismus, welcher es gestattet priviligierte Rechte zu erhalten. Auf der untersten Ebene des Systems interagiert es mit einer Datenbank als Akteur. Die zwei Hauptanforderungen sind Daten zur Verfügung zu stellen und Daten zu persistieren.

Abbildung 1: Anwendungsfall Diagramm



Aus den groben Anforderungen im Use-Case Diagramm kann man eine Reihe von genauen Anforderungen ableiten. Diese sind in 1 beschrieben.

Tabelle 1: Anforderungstabelle - Evaluationsbeispiel

FA 1.1	Das System muss die Erstellung einer Person ermöglichen. Die Person kann dabei einen Dozenten eines Seminares, oder einen Teilnehmer darstellen .
FA 1.2	Das System muss Seminare erzeugen können. Ein Seminar soll mehrere Veranstaltungen haben.
FA 1.3	Es soll möglich sein einen Raum für eine Veranstaltung zu buchen. Eine Buchung gilt für genau einen Raum. Es ist möglich einen Raum mehrmals zu buchen. Die Buchungszeiten dürfen sich nur nicht überschneiden.
FA 2	Das System soll unterschiedliche Nutzer mit unterschiedlichen Rechten zur Verfügung stellen. Es soll Benutzer mit reinen Leserechten geben und Benutzer, die Daten bearbeiten können
FA 2.1	Es soll einen Benutzer-Login geben. Nach dem Login erhält man Schreiberechte
FA 2.2	Die Login-Daten des Systems sollen in einer externen Properties Datei gespeichert werden
FA 2.3	Die Passwörter des Systems sollen über das AES Verfahren verschlüssel werden, bevor sie abgelegt werden.
FA 3.1	Der Verwalter des Systems kann neue Räume zum System hinzufügen/-löschen und diese bearbeiten. Ein Raum kann nicht zwei Mal existieren. Der Name muss eindeutig sein.

FA 3.2	Der Verwalter des Systems kann Seminare verwalten, wobei ein Seminar aus unterschiedlichen Veranstaltungen bestehen kann.
FA 3.3	Es soll möglich sein eine Buchung zu bearbeiten bzw. zu verlegen.
FA 4	Die Daten der Anwendung sollen in einer Datenbank abgelegt werden
FA 4.1	Die Datenbank muss Daten speichern und zur Verfügung stellen können.
FA 5	Das System soll eine grafische Benutzeroberfläche besitzen
FA 5.1	Die grafische Benutzeroberfläche wird durch Objekttemplates im XHTML Format beschrieben.
NFA 1	Die Antwortzeiten des Systems dürfen nicht länger als 5 Sekunden sein.
NFA 2	Das System soll auf allen Systemen lauffähig sein, die einen Internet-Browser besitzen.
NFA 3	Das UI des Systems soll intuitiv bedienbar sein
NFA 4	Die Veränderung des Datenmodels für den Anwendungsfall soll ohne großen Implementierungs Aufwand möglich sein
NFA 4.1	Das System soll eine leichte Änderung der Frontend-Templates ermöglichen. Dies soll über intutitve JSF Templates ermöglicht werden

2.3 Datenmodell der Referenz-Anwendung

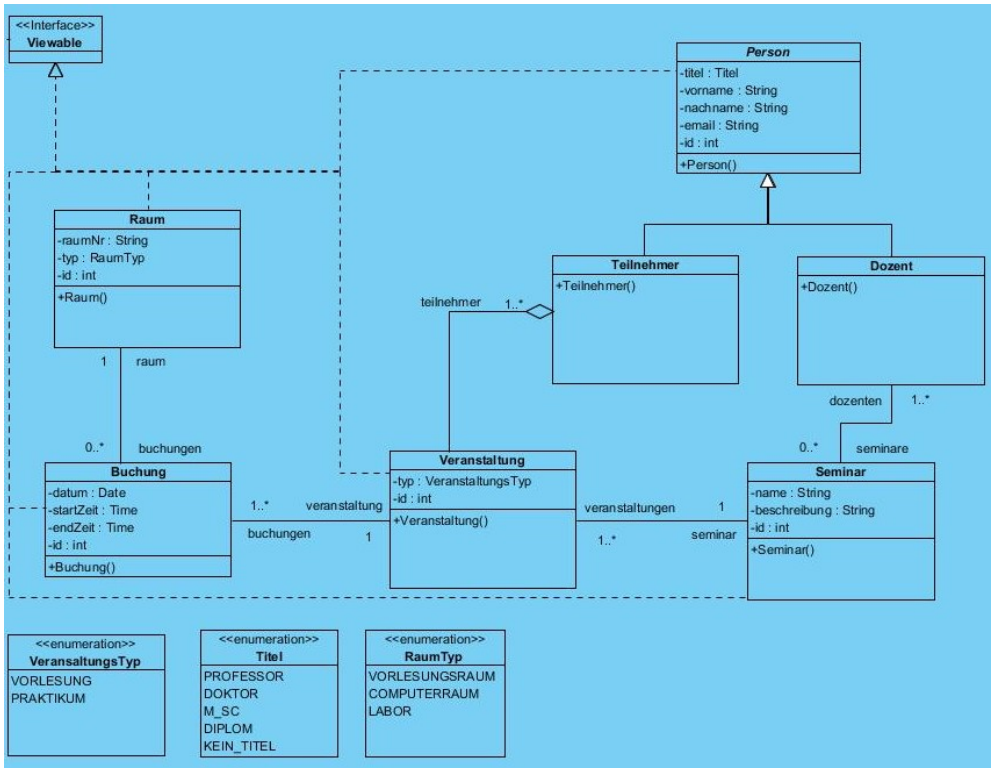
Das aus den Anforderungen entstandene Datenmodell ist in 2 abgebildet. Die Tabelle 2 beschreibt alle Klassen und deren Abhängigkeiten.

Tabelle 2: Klassenbeschreibung

Viewable	Erweitert das *Viewable* Interface des GUI Paketes. Es fügt keine Funktionalität hinzu und dient lediglich der Verminderung einer Bindungsstärke zwischen den Paketen. Die eigentliche Schnittstelle wird in **??** beschrieben.
Person, **Teilnehmer**, **Dozent**	Die Klasse Person, stellt die Überklasse der Teilnehmer und Dozenten dar. Ein Dozent ist dabei spezifisch die Person, welche ein Seminar leitet. Er kennt alle seine Seminare. Ein Teilnehmer ist ein Zuhörer des Seminares, dabei kann ein Teilnehmer wiederum ein Dozent in einer anderen Rolle sein.

Seminar	Die Klasse Seminar repräsentiert ein Modul eines Semesters. Dabei kann ein Seminar mehrere Veranstaltungen haben.
Veranstaltung	Eine Veranstaltung gehört unmittelbar zu einem Seminar. In unserem Beispiel kann sie entweder eine Vorlesung oder ein Praktikum sein. Dies wird durch das Enum VeranstaltungsTyp bestimmt. Eine Veranstaltung muss mindestens eine Buchung enthalten, damit sie irgendwo stattfinden kann.
Buchung	Eine Buchung stellt die Reservierung eines Raum durch eine Veranstaltung dar. Eine Buchung gehört immer genau zu einem Raum und genau einer Veranstaltung.
Raum	Ein Raum repräsentiert einen tatsächlich buchbaren Raum einer Bildungseinrichtung. Der Raum kann mehrfach gebucht werden. Es darf nur keine Zeitüberschreitungen innerhalb der Buchungen geben.
VeranstaltungsTyp, **Titel**, **RaumTyp**	Enums, die innerhalb des Systems verwendet werden. Eine Veranstaltung kann vom Typ Vorlesung oder Praktikum sein. Eine Person keinen einen Titel besitzen, wobei nur eine beschränkte Auswahl von uns angeboten wird. Ein Raum kann einen bestimmten Raumtypen haben, den man auswählen möchte.

Abbildung 2: Datenmodell Klassendiagramm



3 Gesamtsystem

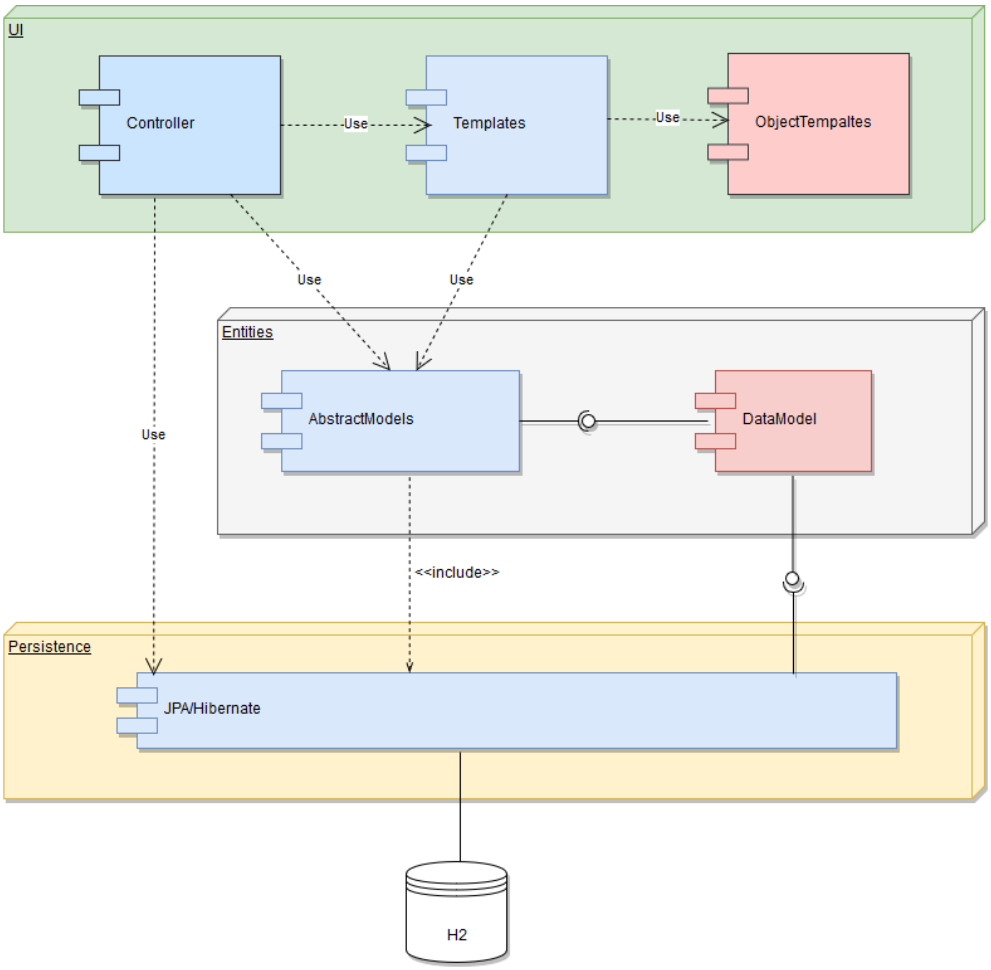
3.1 Komponentendiagramm

In der Abbildung 3 ist die Übersicht der Systembausteine dargesstellt. Das System selbst ist in drei Schichten aufgeteilt. Eine Schicht, für die Benutzeroberfläche, die Steuereinheiten und Oberflächenelemente enthält. Eine Schicht für das Datenmodell und eine Schicht für die Daten-Persistenz.

Innerhalb der Schichten, gibt es zwei rot markierte Komponenten. Die „ObjectTemplate-Komponente“ und die „DataModel-Komponente“. Diese stellen die Elemente dar, die an den spezifischen Anwendungsfall angepasst werden müssen. Die „Datamodel“-Komponente entspricht dem tatsächlichen Datenmodel der Anwendung. Alle Elemente, die vom Framework dargestellt werden sollen, müssen das Viewable-Interface aus der „AbstractModel“-Komponente implementieren. Zusätlich müssen für alle Elemente des Datenmodells eigenene Frontend-Template erstellt werden, welche ein bestimmtes Namensschema befolgen - Die „Object-Templates“. Ein einheitliches Namensschema löst das Problem mit unterschiedlichen Darstellungsmöglichkeiten. Innerhalb der Templates ist es möglich Validierungskriterien für die Element-Attribute zu formulieren, wodurch die Konsistenz des Datenbestandes gesichert werden soll.

Die Steurungselemente des User-Interfaces, sowie die Templates, arbeiten mit den Elementen der „AbstractModel“-Komponente. Dies ermöglicht einen Austausch des Datenmodels der Anwendung. Damit der Datenbestand persistiert werden kann, setzt das Framework voraus, dass jedes Viewable-Element eine Entity des JPA-Frameworks ist. Das „Loader“-Element der „Controller“-Komponente greift direkt auf das JPA zu, um die Daten in der Datenbank zu verwalten. Es arbeitet jedoch immer nur mit den „Viewable“-Elementen.

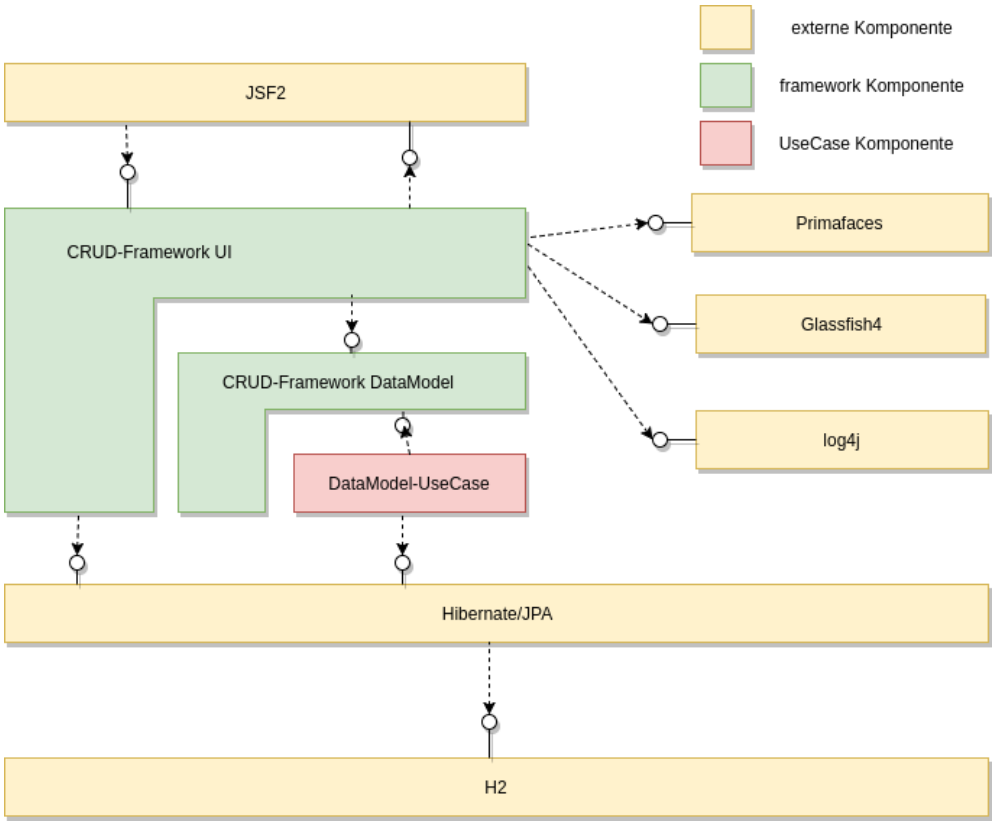
Abbildung 3: Komponenten Diagramm



Die Abbildung 4 zeigt das Gesamtsystem im Kontext aller extern verwendeten Komponenten. Die Benutzeroberfläche basiert auf der Primefaces Technologie. Der JSF-Ansatz erlaubt ein Template basiertes arbeiten und ermöglicht somit einen hohen Grad der Wiederverwendbarkeit. Es kam die JSF-Erweiterung „Primefaces“ zum Einsatz, welche vorgefertigte UI-Elemente bereitstellt, um einen möglichst schnellen Fortschritt zu ermöglichen. Der JSF-Ansatz ist webbasiert und benötigt deswegen einen Applikation-Server. In dem System wird der Glassfish4-Server eingesetzt. Obwohl der Support abgestellt wurde, bietet er eine sehr gute Referenzimplementierung. Für ein Logging von Ereignissen innerhalb des Systems wurde „log4j“ verwendet.

Auf der untersten Ebene des Systems ist die Datenpersistenz, welche über JPA umgesetzt wurde. Es wird darunter die Datenbank H2, wegen ihrer Schlankheit eingesetzt. Das Datenmodell kommuniziert über die JPA-Annotationen mit der Datenbank. Während es in der UI-Schicht einen Kontroller gibt, welcher die expliziten Datenbankzugriffe und die Kommunikation mit der Benutzerobefläche regelt.

Abbildung 4: konkrete externe Komponenten



4 Systembechreibung des CRUD-Frameworks

4.1 Beschreibung und Lösungsstrategie

Die Grundidee hinter dem Framework basiert auf dem Gedanken, dass alle simplen CRUD-Anwendungen ein gleiches Verhalten aufweisen, da sie die selbe Funktionalität zur Verfügung stellen, nämlich *Lesen, Schreiben, Löschen und Ändern*. Wenn man eine weitestgehend individuelle Benutzeroberfläche außer Acht lässt, liegt der einzige Unterschied zwischen den Anwendungen im Inhalt und der Bedeutung der Datenmodelle. Darauf basierend braucht es auch an das Datenmodel angepasste Darstellungen in der Benutzeroberfläche. Das CRUD-Framework geht davon aus, dass ein Datenmodel einer CRUD-Anwendung aus drei Komponenttypen besteht. Die „Viewable"-Komponenten, die im Normalfall den Klassen des Datenmodels und den JPA-Entities. Die „ModelLists", die Sammlungen von Viewable-Komponenten darstellen. Die „Associations", die eine Erweiterung der ModelList sind. Sie bilden die Beziehungen zwischen den einzelnen Datenmodel-Elementen.

Diese abstrakte Darsellung der Datenmodel-Elemente ist ausreichend um die gesamte Funktionalität einer CRUD-Anwendung abbilden zu können. Diese Eigenschaft macht sich das CRUD-Framework zu nutze. Ein Anwender muss nur noch die definierte Schnittstelle in sein Datenmodel implementieren und passende XHTML-Temlates für die Darstellung der Elemente erstellen.

4.2 Anforderungen

Das Framework muss in der Lage sein ein Datenmodell und dazugehörige Templates im XTHML Format entgegen zunehmen und eine funktionierende CRUD-Applikation daraus zu machen. Das Framework soll die gesamte Steuerung der Applikation übernehmen, indem es eine Reihe von Steuereinheiten und Interfaces zur Verfügung stellt. Einer der Hauptansprüche des Systems, ist eine automatische Erkennung der Relationen zwischen den Datenmodellen und die richtige Abbildung derer. Desweiteren sind Hauptanforderungen an das Framework eine Schnittstelle für Fehlermeldungen und Logging. Die Daten werden vom System mittels JPA persistiert, sofern das Datenmodell JPA-Entitäten implementiert.

Tabelle 3: Anforderungen CRUD Framework

FA 1.1	Das System muss die Integration von nahezu beliebigen Datenmodellen ermöglichen.
FA 1.2	Das System muss nach der Integration, die Erstellung von neuen Elementen des Datenmodels ermöglichen.
FA 1.3	Das System muss die gängigen Create Read Update Delete-Operationen ermöglichen
FA 2	Das System muss in der Lage sein Relationen zwischen den Elementen des Datenmodels darstellen zu können
FA 3.1	Die Objektansichten müssen für den Anwendungsfall angepasst agepast werden können
FA 3.2	Innerhalb der Templates wird grundsätzlich die Variable „object“ verwendet
FA 4	Das System muss in der Lage sein Systemifromationen und Fehlermeldungen in der UI anzuzeigen
FA 5	Das System muss die Benutzerauthentifizierung ermöglichen
FA 6.1	Das System soll durch den Ansatz des JPA die Datenpersistenz sicher stellen
FA 6.2	Das System soll einen Austausch der Datenquelle ermöglichen
NFA 1	Die Antwortzeiten des Systems dürfen nicht länger als 5 Sekunden sein.
NFA 2	Das System soll auf allen Systemen lauffähig sein, die einen Internet-Browser besitzen.
NFA 3	Die UI des Systems soll intuitiv bedienbar sein
NFA 4	Die Veränderung des Datenmodels für den Anwendungsfall soll ohne großen implementierungs Aufwand möglich sein
NFA 5	Die Systemarchitecktur soll dokumentiert werden

4.3 Datenmodel des Frameworks

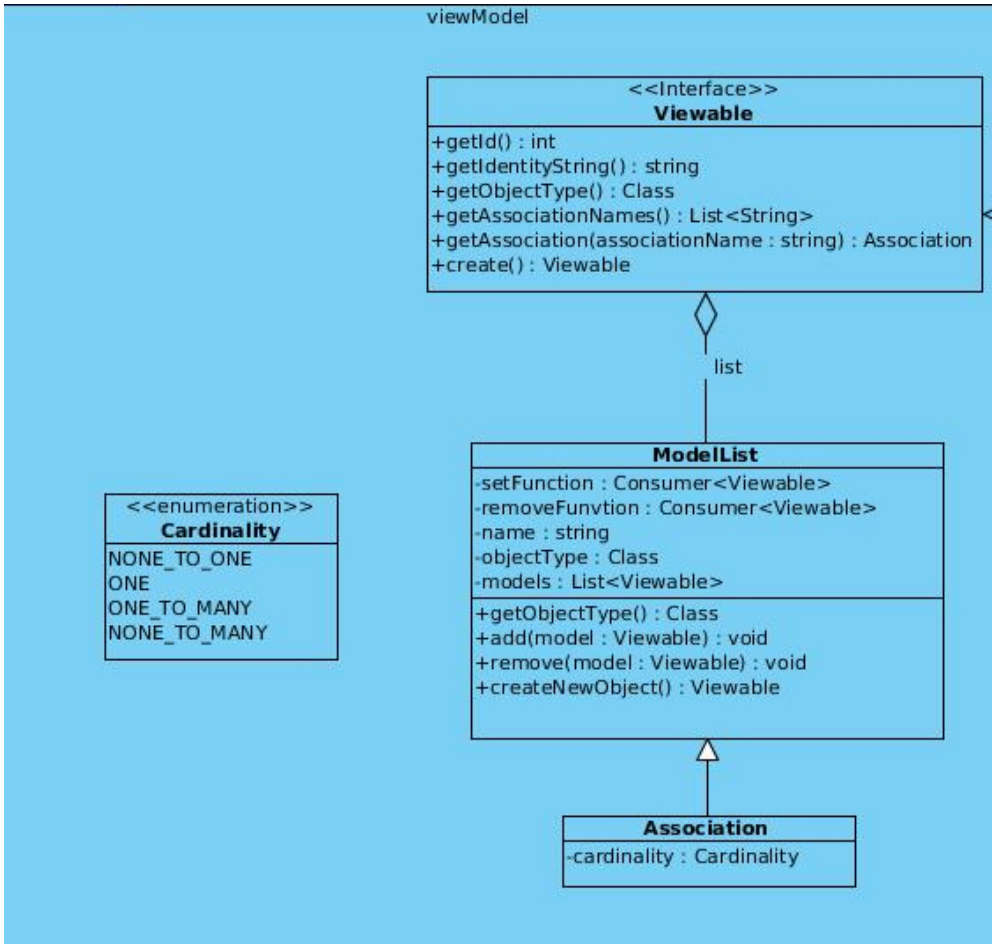
In den vorhergehenden Kapiteln wurden das Datenmodel des Frameworks bereits erläutert. Die Datenstruktur ist in der Abbildung 5 dargesstelt. Dieses besteht aus drei Elementen:

Tabelle 4: Klassenbeschreibung

Viewable Interface	Diese Schnittstelle muss von jeder Klasse des Datenmodels implementiert werden, die in dem Userinterface dargestellt werden soll. Die Funktionen liefern die Namen der Klassen, die Viewable-Elemente repräsentieren, sowie die Beziehungs-Informationen zwischen den Elementen.
ModelList	Stellt eine Sammlung von Datamodel-Elementen dar.

Association	Ist eine Erweiterung der ModelList. Die Objekte dieser Klasse repräsentieren die Beziehungen zwischen den Datenmodel-Elementen. Aus diesem Grund beinhalten diese Objekte Informationen über die Kordinalität der *Association*, eine Referenz auf das Associationsende und Zeiger auf die Funktionen, die zum hinzufügen bzw. entfernen von Elementen aus der Assoziation verwendet werden.
Cardinality	Ein Enum für die unterschiedlichen Kardinalitäten, die ein Assoziationsende haben kann.

Abbildung 5: GUI-Modell Klassendiagramm



4.4 Templates

Als Templates werden die XHTML-Vorlagen bezeichnet. Diese können in zwei Gruppen aufgeteil werden: Object-Templats und Service-Templates. Die Object-Templates werden für die Darstellung der einzelnen Viewable-Objekte verwendet. Diese müssen einem bestimmten Namenschema folgen, welches den Klassenname beinhaltet. Der Inhalt der Objekt-Templates kann abhängig vom Anwendungsfall angepasst werden. Die einzelnen Elemente dieser Templates können zusätliche Validierungskriterien beinhalten.

Die Templates sind durch Kompositionen miteinander verknüpft. Die Applikation beginnt in der Hauptansicht dem Index-Template, von welcher man in vier Listenansichten gelangt. In einer Listenansicht, kann man ein neues Objekt erzeugen, ein vorhandenes Objekt bearbeiten bzw. löschen, oder einfach einsehen. Wenn man ein Objekt löscht, so bleibt man in der Listenansicht - In jedem anderen Fall gelangt, man in die Objektansicht. Jede Objektansicht besitzt Referenzen auf andere Objekte, also Assoziationen. Diese Referenzen, werden in der Objektansicht ebenfalls gezeigt. Wenn man diese Referenz einsehen will, so gelangt man immer zunächst in die Listenansicht der angeforderten Objekte, auch wenn nur ein Objekt vorhanden ist.
Die Service-Templates verwenden nur die Objekte der „AbstractModel"-Komponente und können nach Bedarf Object-Templates einbinden. Es gibt folgende Service-Templates:

Index-Template Ist das Template für die Hauptseite der Benutzeroberfläche. Es beinhaltet die „Sidebar" der Anwendung. Zusätzlich enthält es das „Contentpane" der Anwendung, in dem andere Templates eingefügt werden können. In Abbildung 6 sieht man eine Repräsentation der Hauptansicht der integrierten Listenansicht und einer Objektansicht.

ListView Ist das Template für die Listenansichten. Es wird für die Darstellung der ModelList bzw. Association verwendet. Eine ListView beinhaltet immer zugehörige Object-Templates, da man einzelne Objekte innerhalb der Listenansicht ebenfalls ansehen kann.

NewView Das Tempalte wird bei der Ertellug eines neuen Objekts verwendet. Es bindet die zugehörigen Object-Templates sowie das SelectView-Template ein.

SelectView Dieses Template wird verwendet wenn existierende Objekte zu einer Assoziation hizugefügt werden sollen. Ein Beispiel sieht man in Abbildung 8

LogInView Ein Template, welches für die Benutzerauthentifizierung verwendet wird. Die dazugehörige Umsetzung sieht man in Abbildung 7

Abbildung 6: Templates Screenshot

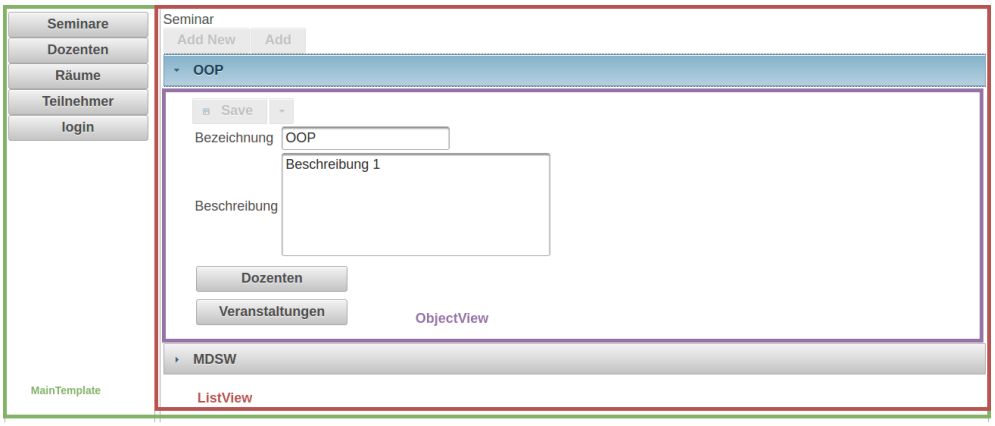


Abbildung 7: LoginView Screenshot

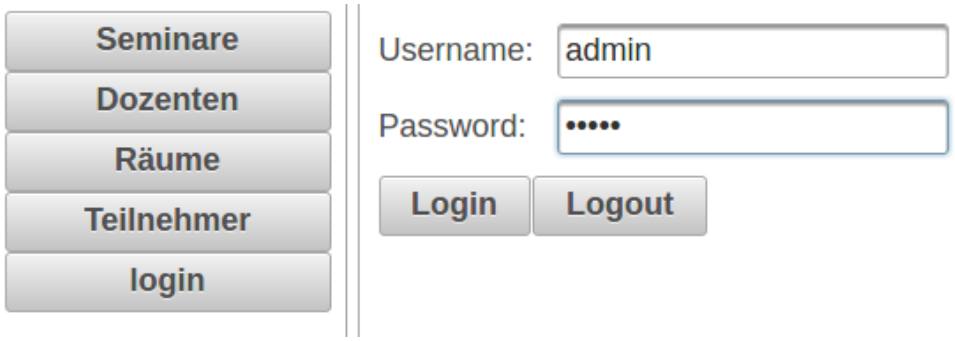
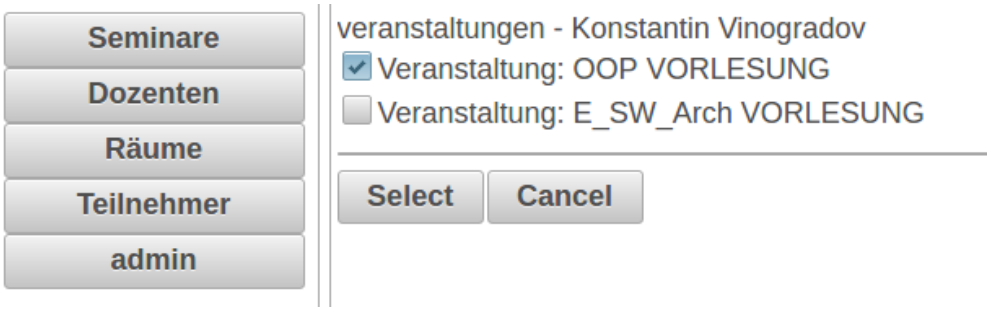


Abbildung 8: SelectView Screenshot



4.5 Controller

Die Controller-Komponenten können, mit der Ausnahme des MainController, in zwei Gruppen aufgeteilt werden: UI- und Service-Controller. Die UI-Controller sind für die Funktionalität einzelner Ansichten zuständig. Die Serivce-Controller regeln den Datenbankzugriff, Fehlerbehandlung und Logging.

4.5.1 MainController

Der MainController nimmt eine Sonderrolle ein. Er dient als Kommunikations-Knotenpunkt und bestimmt welcher UI-Controller die Steuerung übernehmen soll. Dies ist die

Hauptsteuereinheit, weswegen sie alle anderen Steuereinheiten kennt und deligiert. Zusätzlich steuert sie, welches Template gerade sichtbar ist.

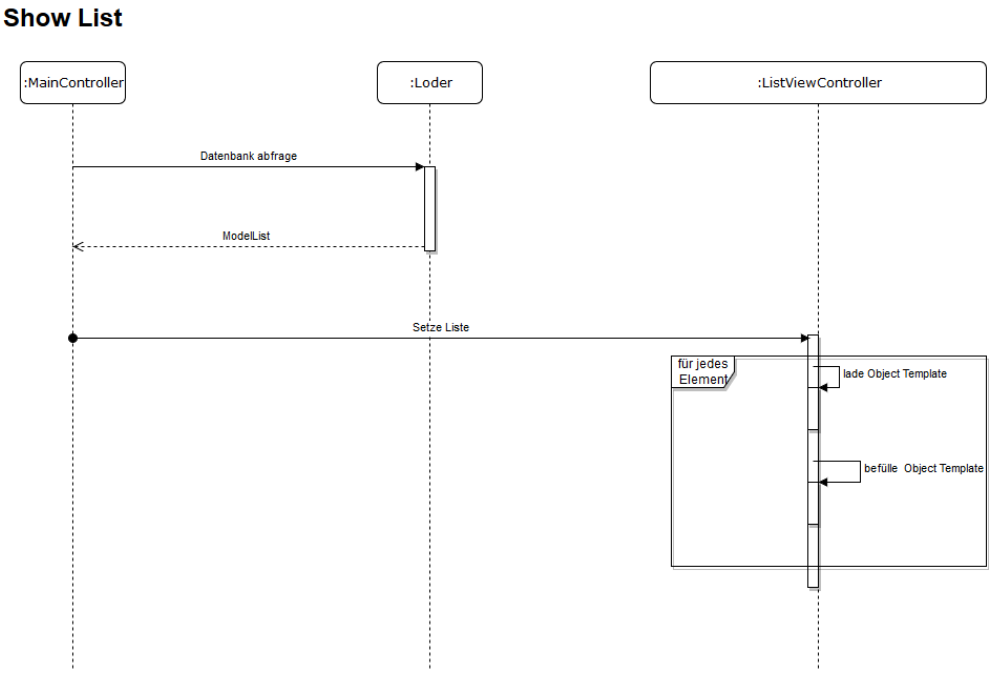
4.5.2 UI-Controller

Diese Steuereinheiten sind rein für die Verwaltung und Kommunikation der Benutzeroberfläche verantwortlich. Die drei wichtigsten Steuereinheiten sind der „ListViewController“, „SelectViewController“ und „NewViewController“, die ausführlicher erläutert werden.

ListViewController Der ListViewController ist für die Verwaltung der Listenansicht zuständig. Der MainController überträgt ihm die Kontrolle nachdem mit dem Loaderkommuniziert wurde, um die benötigten Daten aus der Datenbank zu laden.

Der typische Ablauf für diese Steuereinheit ist das Anzeigen einer Liste. Der Ablauf bei der Erstellung der Listenansicht ist in dem Diagramm 9 dargestellt. Dabei bindet das ListView-Template zugehörige Objekt-Templates zur Laufzeit ein. Die Datenobjekte und die Information, ob der Nutzer die Daten bearbeiten darf, werden bei der Anbindung übergeben. Eine Typprüfung der Datenobjekte findet nicht statt.

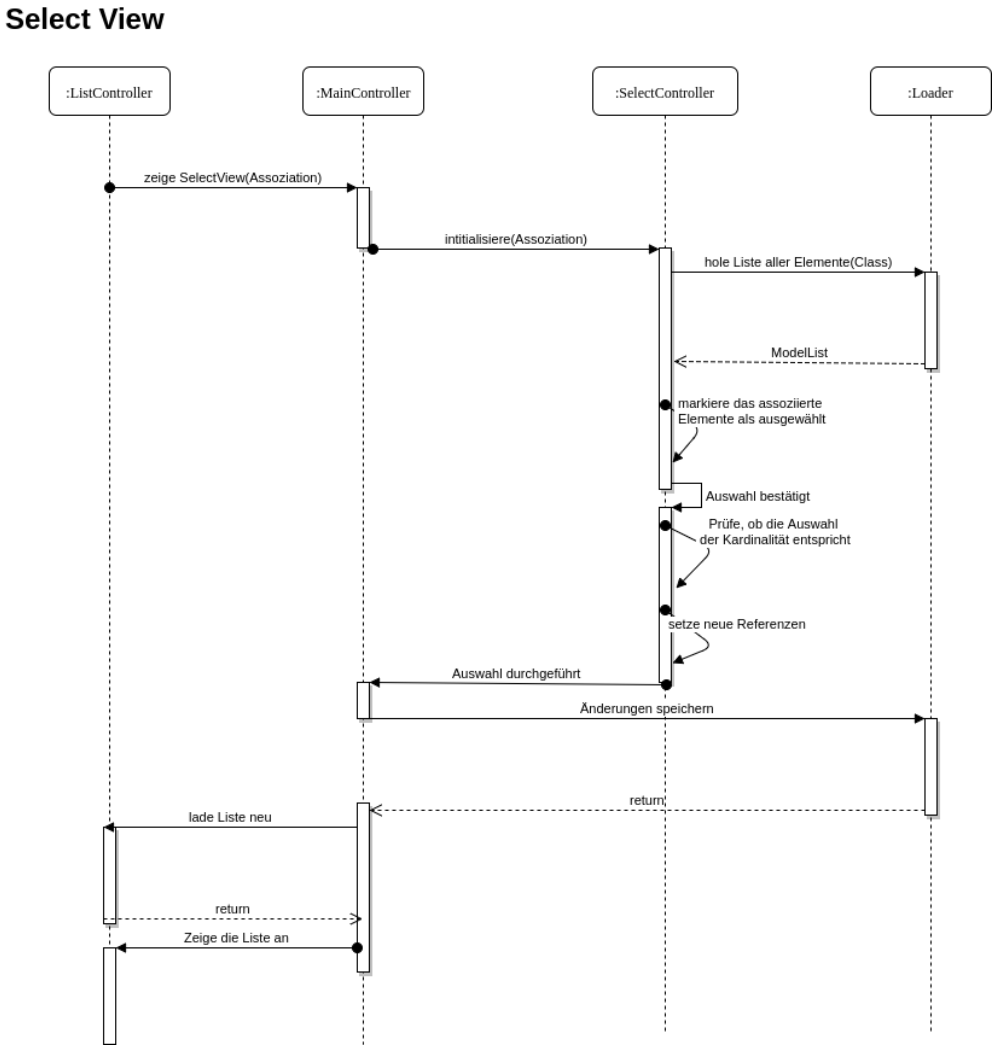
Abbildung 9: ShowList Sequenzdiagramm



SelectViewController Der SelectViewController ist für die Zuordnung von existierenden Objekten zu anderen Objekten zuständig. Er ermöglicht z.B., dass in diesem generischen Ansatz ein Teilnehmer einer Veranstaltung zugeordnet werden kann.

Die Select Funktionalität wird von der Listenasicht zur Verfügung gestellt. Sie wird nur für die Assoziationsdarstellung benötigt. Der Ablauf ist in dem Diagramm 10 dargestellt. Zu vermerken ist, dass die Validerung der Auswahl anhand der Kordinalität seiner Assoziation, nicht unter die Aufgaben des SelectViewControllers fällt.

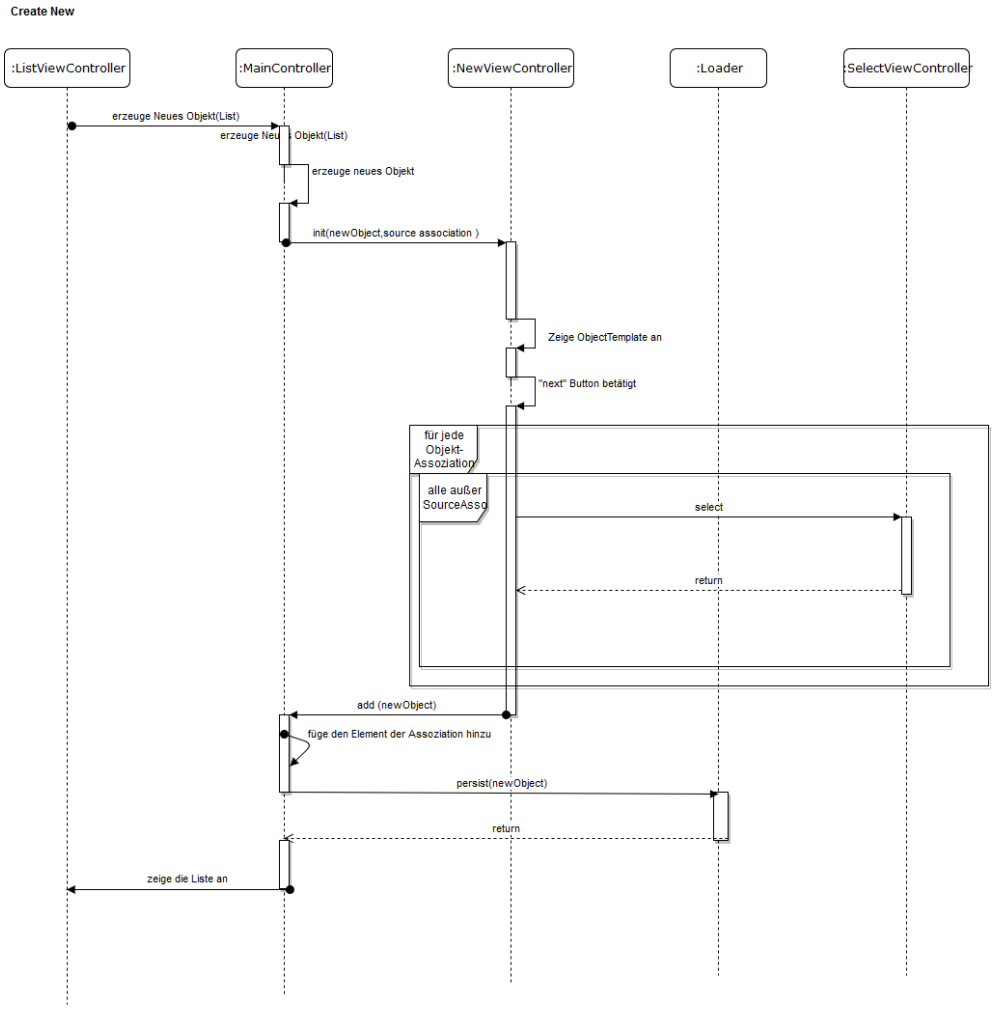
Abbildung 10: Select Sequenzdiagramm



NewViewController Der NewViewController ist für die korrekte Erzeugung und Zuordung eines neu angelegten Objektes und seiner Abhängigkeiten verantwortlich.

Das Diagramm 11 stellt die Kommunikation zwischen den Controller-Elementen bei der Erstellung eines neun Datenobjekts dar. Der Vorgang kann aus der ListView ausgelöst werden. Der Ablauf unterscheidet sich abhängig davon, ob es sich bei dem Ausgangselement um eine „ModelList“ oder eine „Association“ handelt. Bei einer Association kann das Datenobjekt automatisch bei der Erstellung gestzt werden.

Abbildung 11: CreateNewView Sequenzdiagramm



UserLoginController Der UserLoginController regelt die Zugriffsrechte im System und ist für den gesamten Login-Vorgang verantwortlich.

Wenn ein Benutzer sich am System anmelden will, so gibt er seine Nutzerdaten ein. Diese Steuereinheit sucht nach dem Nutzernamen und entschlüsselt das AES verschlüsselte Passwort, um es mit der Eingabe abzugleichen. Sollte der Vorgang erfolgreich sein, wird ein globaler Flag umgeswitcht, welcher die Bearbeitungsfunktionalität freischält.

4.5.3 Service-Controller

Die Service-Controller werden von anderen Controllern verwendet, werden aber nicht direkt aus der UI angesprochen. Es gibt die folgenden zwei Service-Controller:

MessagesViewController Der MessagesViewController ist ein Wrapper für unsere Logging-Komponente „log4J“. Es bietet die Möglichkeit alle Ereignisse an zentraler Stelle zu verwalten und ermöglicht gleichzeitig die Kommunikation mit dem Nutzer, über diverse Anzeigen, die Primefaces zur Verfügung stellt.

Loader Der Loader übernimmt die Kommunikation mit der Datenbank. Er hält einen EntityManger und CRUD-Funktionalität für die Datenbank. Zusätzlich enthält diese Steuereinheit den Datenbank-Treiber, welcher benötigt wird, da wir uns für eine lokale Datenbank entschieden haben und JPA dann keine automatische Konfiguration vornehmen kann.

5 Entwurfsentscheidungen

In diesem Kapitel werden die wichtigsten Architekturentscheidungen kurz erläutert.

Tabelle 5: Entwurfsentscheidungen

Primefaces	Eine der Vorgaben war es mit JSF zu arbeiten. Primefaces ist eine Erweiterung der Java Server Faces um UI-Elemente und Valiedierungsmechanismen.
Maven	Maven ist heutzutage ein sehr gängiges und auch praktisches Werkzeug um Projekt-Abhängigkeiten dynamisch und automatisch herunterzuladen und einzubinden.
SinglePage Applikation	Der Kompositionslastige Aufbau einer CRUD-Anwendung hat die Entscheidung ein Stück weit abgenommen, da die einzelnen Templates ebenfalls nur als Komposition in immer allgemeineren Templates sind. Zusätzlich ist eine solche Ansicht intuitiv und benutzerfrundlich.
JPA	Object Relational Mapping (ORM) Frameworks sind heutzutage Gang und Gebe. JPA ist hierbei ein Java EE eigenes ORM, welches die Persistierung von Datenbeständen erheblich vereinfacht. Ein Vorteil ist zusätzlich, dass die darunterliegende tatsächliche Datenbank ausgetauscht werden kann. Wir haben uns für eine H2 Datenbank wegen seiner Schlankheit entschieden.
ManagedBean	Wir haben unser System auf die „@Named“ Annotation umgestellt und festgestellt, dass die Ladezeiten des Systems in der Initialisierungsphase bis zu fünf Sekunden in Anspruch nehmen können. Mit hoher Wahrscheinlichkeit, hat es mit irgendeiner Konfiguration unseres Systems zu tun. Allerdings wurden wir nach langem Suchen nicht fündig und haben uns für die ManagedBeans entschieden, die in unserem System eine sehr viel bessere Performance erbringen.

6 Erweiterungsmöglichkeiten

Eine möglich Erweiterung des Frameworks wäre eine Änderung der Schnittstelle. Anstatt wie bisher ein definiertes Interface anzubieten, könnte man die einzelnen Interface Funktionen auch als Annotationen umsetzen. Eine Klasse im Datenmodell müsste dann die Klasse mit einer Annotation versehen, die zeigt, dass diese in der Benutzeroberfläche angezeigt werden soll. Zusätzlich gäbe es eine Annotation, die über den Assoziationen innerhalb der Klasse steht. Die passenden Setter und Getter Methoden, die in der ModelList benötigt werden, wären über Reflection zugänglich. Der Typ der Klasse wäre über Reflection ebenfalls einfach zu erhalten. Diese Erweiterung nimmt dem Nutzer des Frameworks mehr Arbeit ab. Er muss lediglich die Assoziationen und die Klasse mit Annotationen versehen, anstatt Kenntnis über ein Interface zu haben.

Eine weitere Erweiterungsmöglichkeit, ist die automatische Generierung von Templates. Basierend auf dem Quellcode müsste irgendwo hinterlegt sein, wie ein Datentyp interpretiert werden soll. Beispielsweise ist ein Boolean eine Checkbox, ein Datum ein Date-Element und ein String ein Textfeld. Diese Erweiterung ist allerdings recht restriktiv gegenüber der Benutzerobefläche, alternativ könnte man dem Nutzer die Möglichkeit geben über spezielle Annotationen zu bestimmen auf welche Weise ein Attribut abgebildet werden soll. Alledings ist auch dies mit Restrktionen verbunden - Man gewinnt jedoch an Effizienz, da nun nur noch das Datenmodell implementiert werden muss.

7 Fazit

Es sollte eine Applikation basierend auf den gängigen Java EE Features umgesetzt werden. Hierbei wurden die Technologien JPA und JSF eingesetzt. Die grobe Anforderungen war es ein „größeres“ Projekt umzusetzen. Wir haben uns die Gemeinsamkeiten einer CRUD Applikation angesehen und ein kleines Framework geschrieben, dass recht universell auf eine große Menge an Anwendungen passt und somit Zeit erspart.

Java EE hat einige nützlich Features, die es sich durchaus lohnt zu kennen und einmal angewendet zu haben. So ist JSF mit den vorhandenen Erweiterungen eine gute Methode um sehr schnell eine Anwendung mit grafischer Benutzeroberfläche umzusetzen. JPA hingegen ermöglicht eine sehr einfache und vor allem dynamische Methode um Daten zu persistieren.