Dokumentation

im Studiengang

Informatik Master

Praktikumsdokumentation: Enterprise Software Architecture

Referent	:	Prof. Dr. Ulf Schreier
Vorgelegt am	:	30.06.2017
Vorgelegt von	:	Christian Reser 256305 INM, Yasemin Dogan INM

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

JPA Java Persistence API

JSF Java Server Faces

CRUD Create Read Update Delete

1 Einführung und Ziele

Im Rahmen des Praktikums der Lehrveranstaltung „Enterprise Software Architektur“ soll eine Webapplikation entstehen, bei der alle aktuell gängigen Technologien der Java Enterprise Edition eingesetzt werden sollen. Dazu gehören zuallererst die Java Server Faces (JSF), ein Framework zur Entwicklung grafischer Oberflächen in Webanwendungen. JSF kann durch den Einsatz von Primefaces um neue Funktionen erweitert werden. Primefaces bietet eine Demoseite[1] mit allen verfügbaren Komponenten mit implementierung an, wodurch Einsteiger einen schnelleren Einstieg in die Entwicklung finden können. Zur Persistierung der Daten soll die Java Persistance API (JPA) eingesetzt werden. JPA bietet eine Schnittstelle, um Objekte einfacher in eine Datenbank zu Übertragen, ohne viel SQL benutzen zu müssen. Alle Entitäten, die in einer Datenbank persistiert werden müssen, werden mit den entsprechenden Annotations versehen und mit Hilfe eines Entitätenmanagers können Objekte eines solchen Typs mit einer Zeile Code in die Datenbank geschrieben und herausgelesen werden. Für manche Anwendungsfälle soll eine Zugriffskontrolle stattfinden, sodass für die zu entwickelnde Applikation ein Login zur Benutzerauthentifizierung nötig ist.

[1] link zur demoseite

2 Anwendungsdomäne - Wettbewerbsverwaltung

Als Anwendungsdomäne wurde eine Wettbewerbsverwaltung gewählt. Im folgenden wird die Anwend

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.

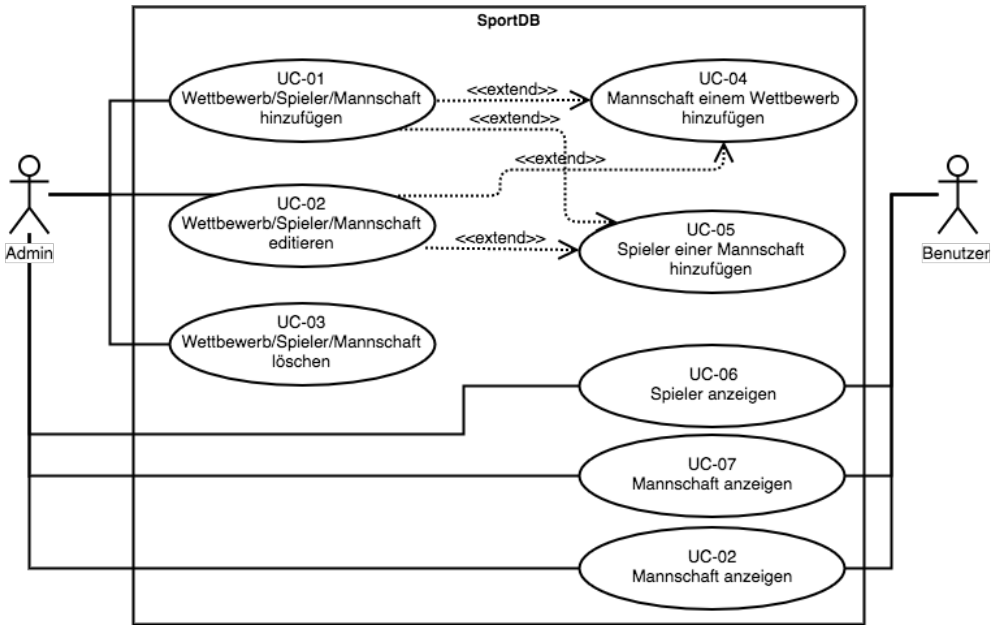
2.1 Beschreibung

Die zu entwickelnde Applikation soll sich nicht auf eine bestimmte Sportart beschränken. Sie soll zum Beispiel für Fussballwettbewerbe genauso einsetzbar sein wie für Handballwettbewerbe oder Volleyballwettbewerbe. Mit Hilfe der CRUD-Operationen(Create, Read, Update, Delete) soll es einem Benutzer möglich sein, Wettbewerbe anzulegen, Wettbewerbe anzuzeigen, Wettbewerbe zu bearbeiten und Wettbewerbe zu löschen. Genauso sollen Mannschaften und ihre zugehörigen Spieler angelegt, angezeigt, bearbeitet und gelöscht werden können. Der Benutzer soll Wettbewerben die Teilnehmenden Mannschaften hinzufügen können, genauso soll er den Mannschaften ihre Spieler hinzufügen können.

2.2 Anforderungen

Die Anforderungen an das System werden in folgendem Use-Case-Diagramm zusammengefasst. Hierbei zeigen sich drei Aktoren. Der Admin kann Datensätze anlegen, editieren, anzeigen und löschen, sowie Mannschaften den Wettbewerben und Spieler den Mannschaften hinzufügen. Der Aktor Benutzer, kann sich lediglich alle Datensätze anzeigen lassen. Als dritter zeigt sich der Datenbank Aktor. Dieser Aktor persistiert die Datensätze und stellt sie zur Verfügung.

Abbildung 1: Anwendungsfall Diagramm

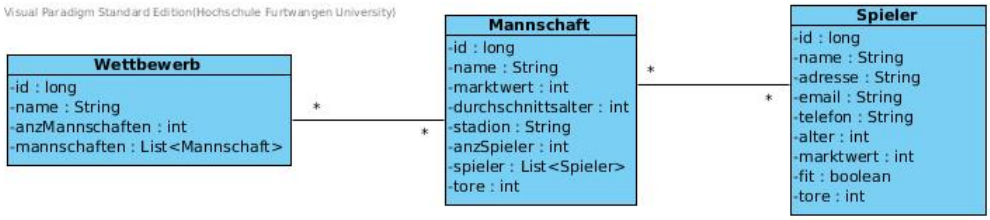


Aus den groben Anforderungen im Use-Case Diagramm kann man eine Reihe von genauen Anforderungen ableiten. Diese sind in **??** beschrieben.

2.3 Datenmodell

Folgendes Datenmodell zu sehen in Abb. 2 ist entstanden. Zu sehen sind die relevanten Modelklassen und ihre Abhängigkeiten zueinander. Diese Daten sind durch das CRUD-Framework modifizierbar. Zwischen Wettbewerb und Mannschaft besteht eine One-to-Many Beziehung, genauso wie zwischen Mannschaft und Spieler.

Abbildung 2: Datenmodell Klassendiagramm



3 Gesamtsystem

3.1 Komponentendiagramm

Abbildung 3 zeigt die fachliche Aufteilung der Softwarekomponenten. Es lassen sich drei Schichten ableiten. An oberster Stelle steht die Webapp mit den Templates aller Seiten und Unterseiten. Diese Templates greifen auf die Controller der Komponenten darunter zu. Jede Komponente, also wettbewerbe, mannschaften, spieler und login beinhalten ihre eigenen Controller in einem Controller Paket. Ausser dem Controller Paket befindet sich noch ein Paket für die Modelklassen, ein Paket für die Producer, welche Objekte erzeugen die injected werden können und ein Servicepaket zur Kommunikation mit der Datenbank jeder Komponente. Die unterste Schicht, die persistance Komponente schreibt und liest Daten aus einer H2 Datenbank mit Hilfe des JPA-Frameworks.

Abbildung 3: Komponenten Diagramm

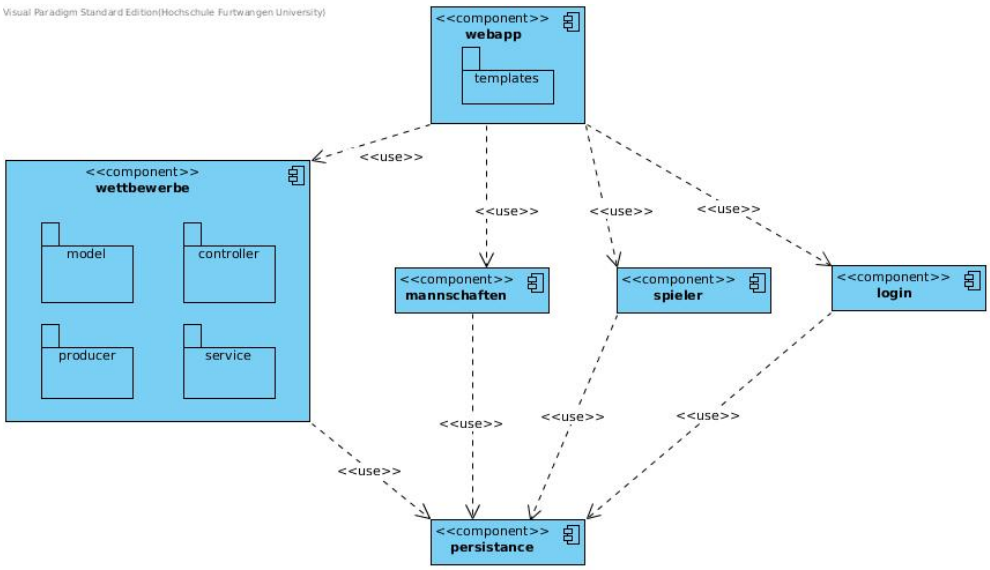
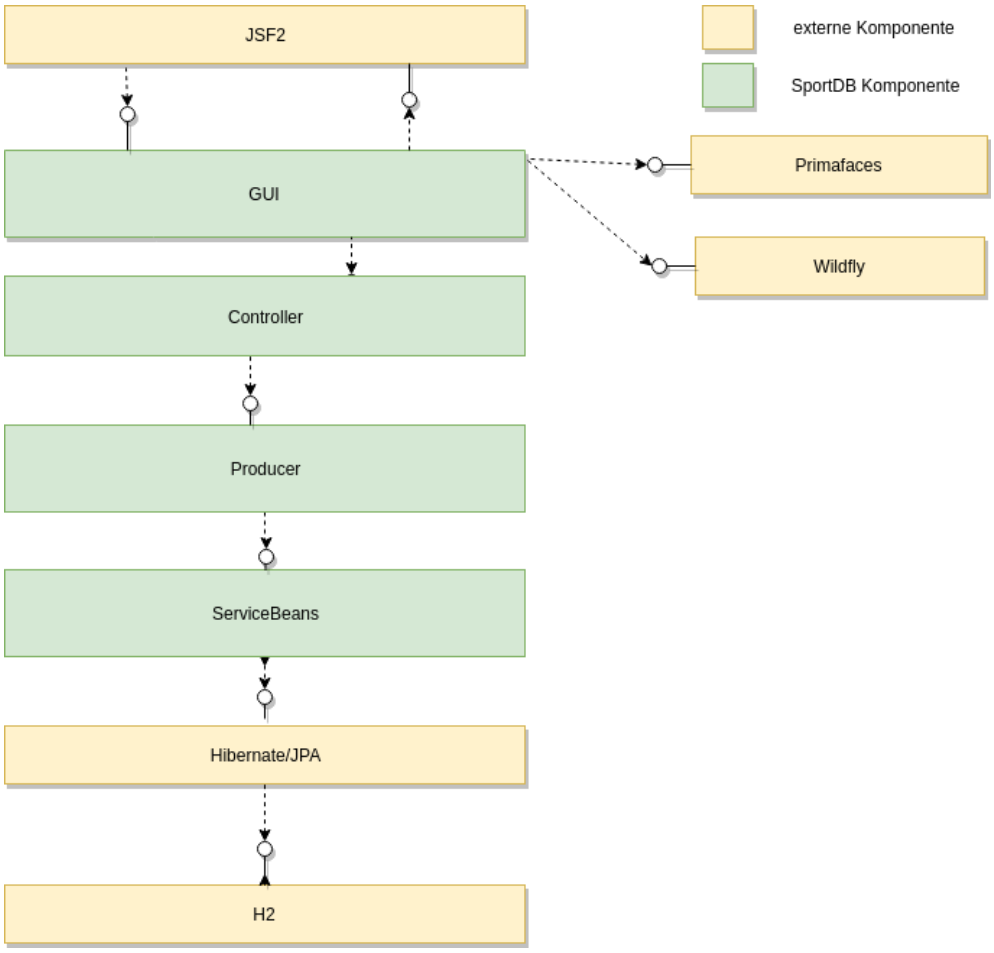


Abbildung 4: konkrete externe Komponenten



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.

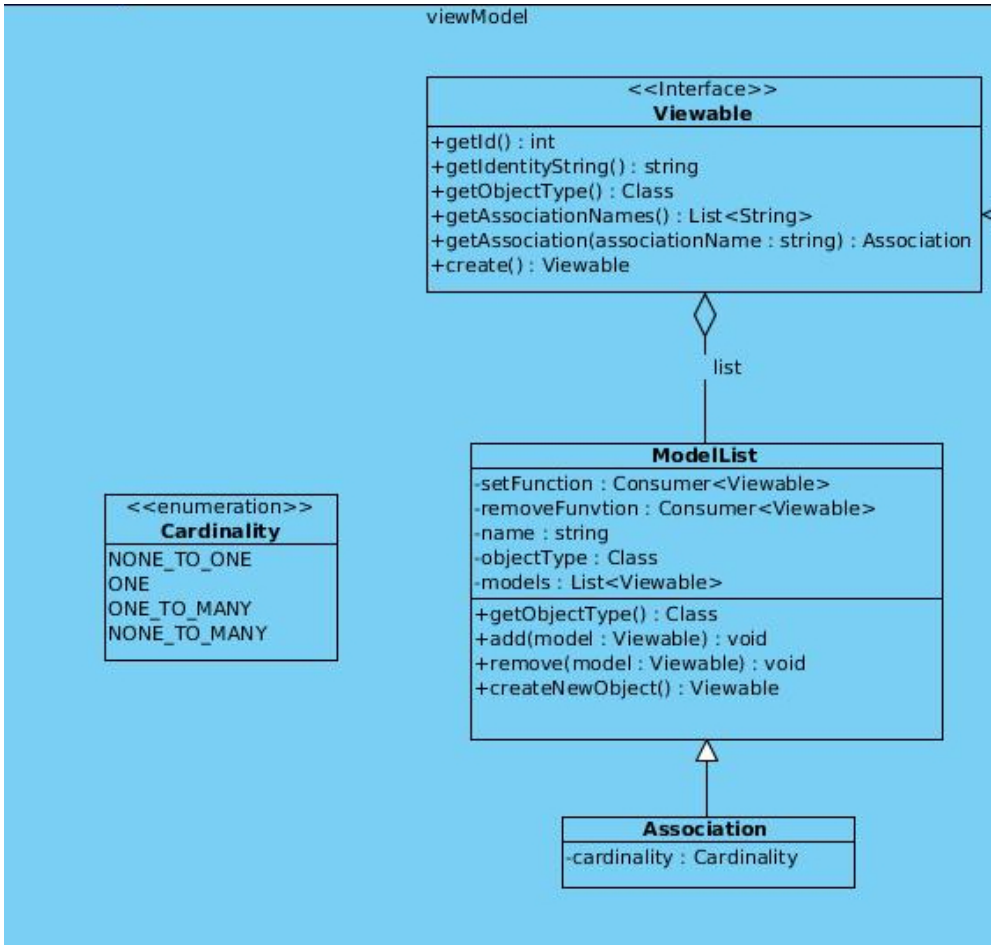
4.2 Anforderungen

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.

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:

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

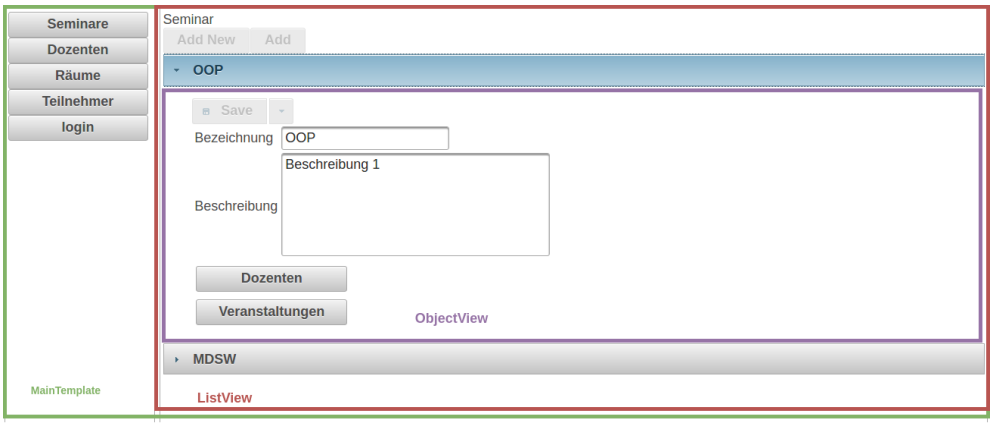


Abbildung 7: LoginView Screenshot

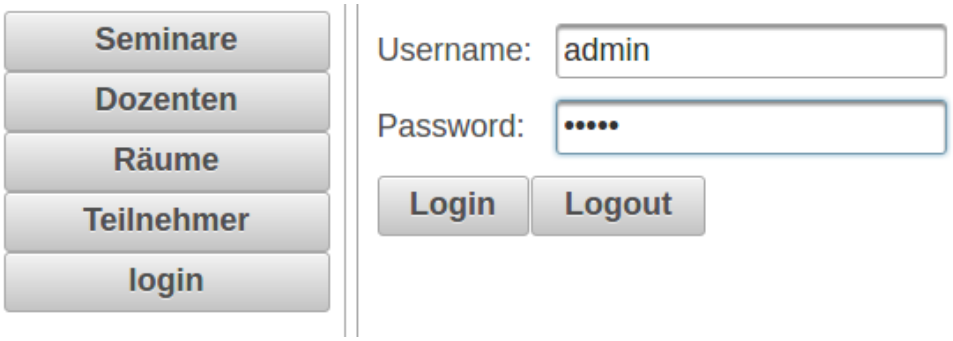
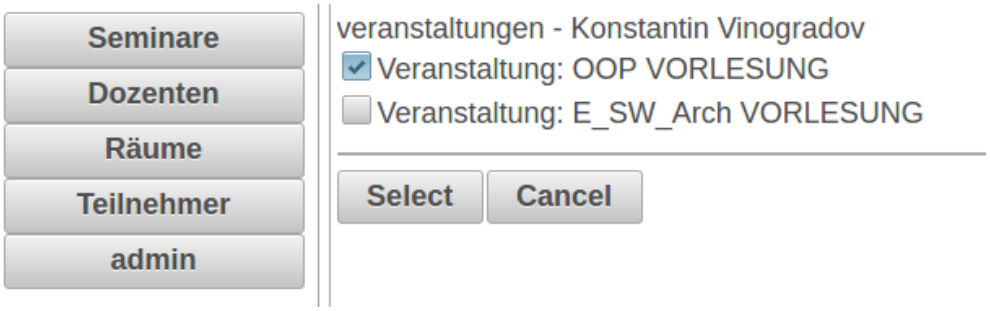


Abbildung 8: SelectView Screenshot



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.

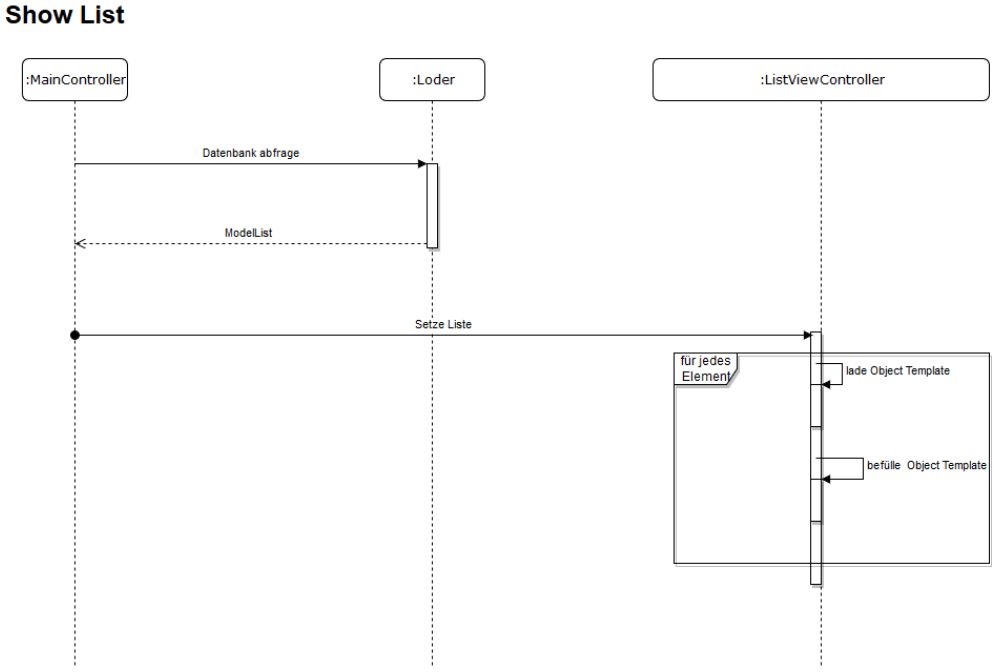
4.5.2 UI-Controller

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.

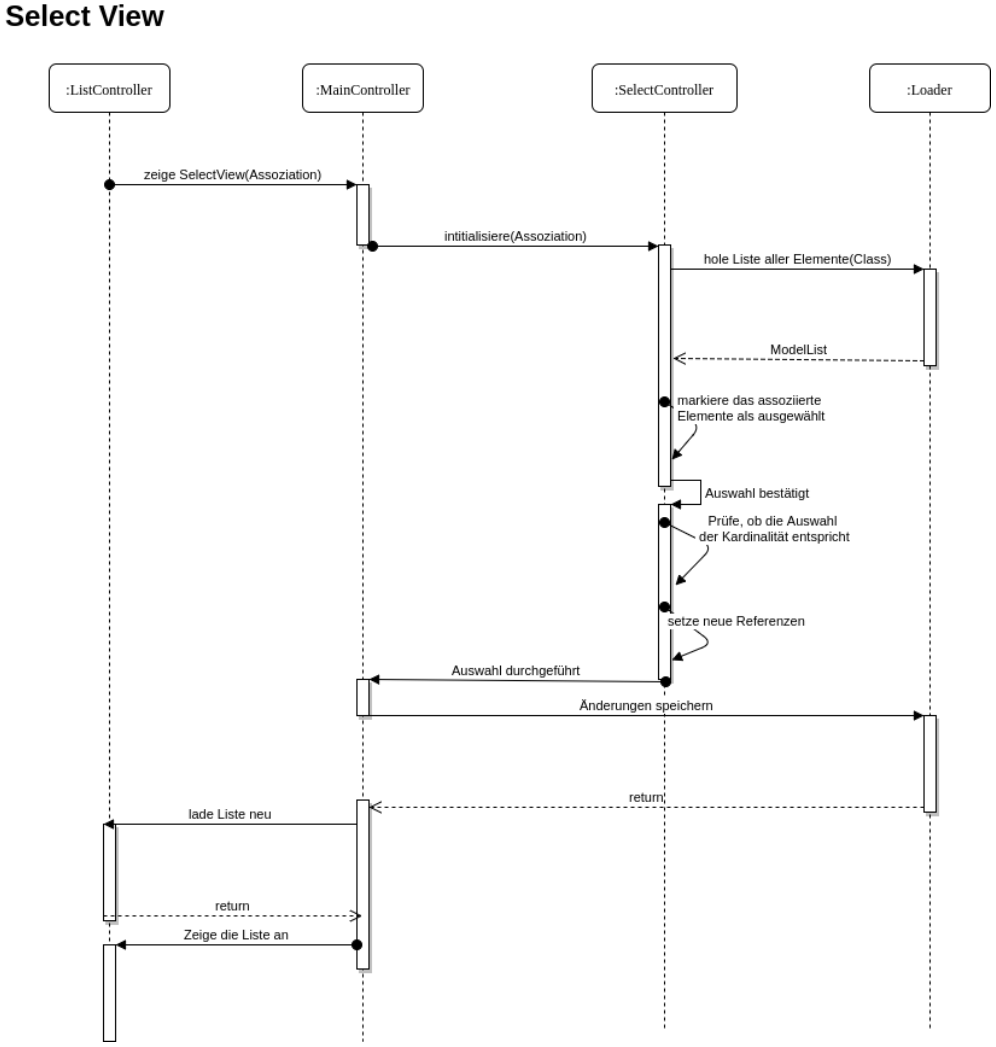
Abbildung 9: ShowList Sequenzdiagramm



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.

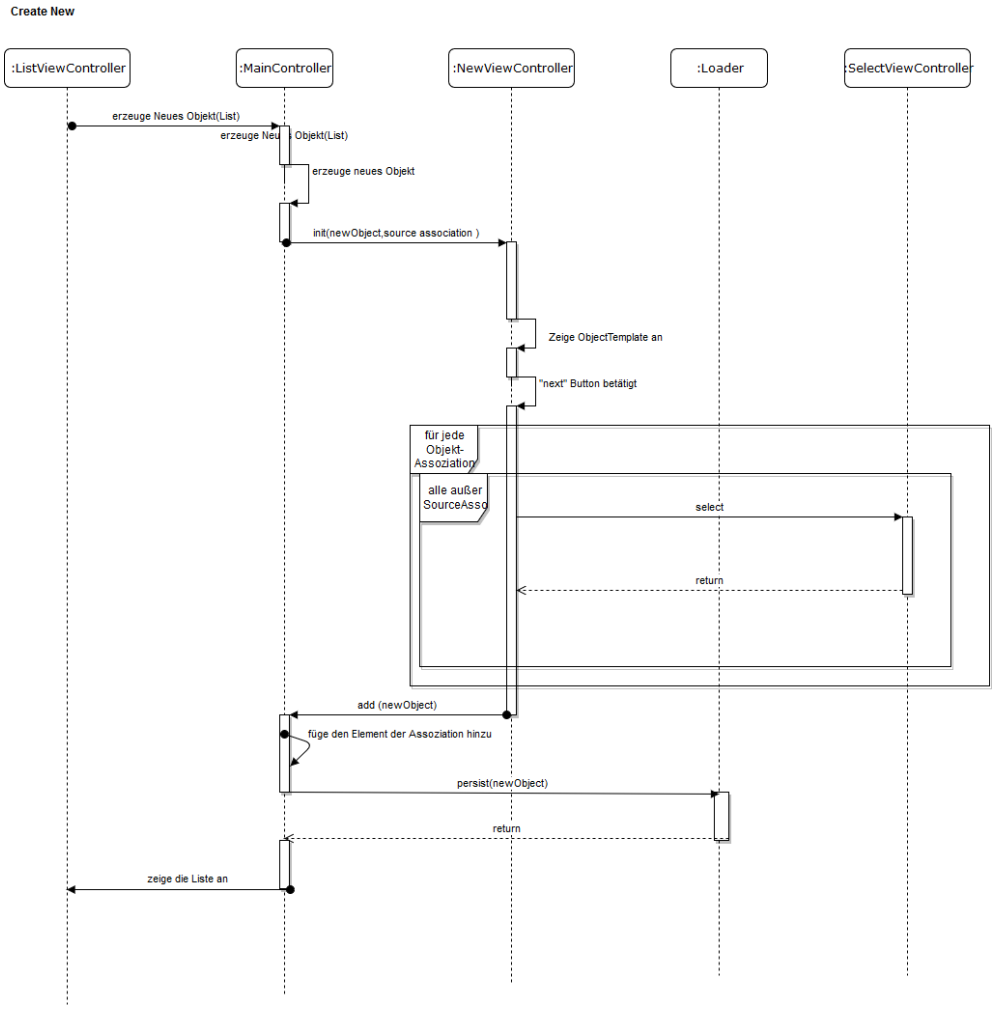
Abbildung 10: Select Sequenzdiagramm



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.

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 Java Persistence API (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.