Universität Bremen
FB 3 – Informatik
Prof. Dr. Rainer Koschke
TutorIn: Marcel Steinbeck

Software–Projekt 2 WiSe 2019/2020

VAK 03-BA-901.02

Architekturbeschreibung
Data Colorado

DataColorado

Liam Hurwitz hurwitz@tzi.de
Kevin Santiago Rodriguez Rey kev_rey@tzi.de
Fabian Kehlenbeck fkehlenb@tzi.de
Aaron Rudkowski rudkowsk@tzi.de
Samuel Nejati Masouleh samnej@tzi.de
Leonard Haddad s_xsipo6@tzi.de

Abgabe: 22. Dezember 2019 — Version 1.0

Inhaltsverzeichnis

Version und Änderungsgeschichte

Die aktuelle Versionsnummer des Dokumentes sollte eindeutig und gut zu identifizieren sein, hier und optimalerweise auf dem Titelblatt.

Version	Datum	Änderungen
0.1	19.09.2019	Erste Schritte
0.2	21.11.2019	Problemkarten und Anwendungsfälle
0.3	06.12.2019	Qualitäts- und Bugfixes
0.4	09.12.2019	Konkrete Problemkarten
0.5	11.12.2019	Ausführungssicht und Datenmodell Entwürfe
0.5.1	12.12.2019	Testen vom Prototyp
0.6	14.12.2019	Zweck in die Einleitung und weitere UML Versuche
0.7	17.12.2019	Modulsicht und Persistenz
0.8	18.12.2019	Evolution und Globale Analyse
0.9	20.12.2019	Korrekturen
0.10	20.12.2019	Alles gemerged
0.11	21.12.2019	Javadoc Sprint
1.0	22.12.2019	finale Abgabe

1 Einführung

Diese Architektur Beschreibung wurde im Rahmen von Software Projekt 2 im Wintersemester 2019/2020 geschrieben. Das Projekt wurde im Rahmen des SFB 1232 Farbige Zustände erstellt. Die SFB-intitative „Farbige Zustände“ erarbeitet sich eine neuartige experimentelle Methode der Werkstoffentwicklung.

Es handelt sich hierbei um eine Webapplikation, welche den Mitarbeitern der SFB-Initiative ermöglichen soll, effizient und ohne Verzögerung durch einen administrativen Mehraufwand Prozessketten für die untersuchten metallischen Konstruktionswerkstoffe zu finden. Außerdem ermöglichen wir es den Forschern der SFB-Initative den Überblick über laufende Prozessketten und deren ProzessSchritte zu behalten. Konkret bedeutet dies, dass wir es den Logistikbeauftragten mithilfe unserer Software ermöglichen, Proben und deren Träger zu verwalten. Den Technologen der SFB-Initiative ermöglichen wir es, den Überblick über ihre Experimentierstationen zu behalten; sie werden immer über die laufenden Prozessschritte informiert sein. Dem Prozesskettenadministrator möchten wir es ermöglichen, Prozessketten zu erstellen und verwalten, welche auf digitalem Wege an die Technologen weitergegeben werden als auszuführende Aufträge, welche diese wiederum annehmen und durchführen können. Außerdem möchten wir den Mitarbeitern im Lager ermöglichen, eine klare Übersicht über die im Lager sich befindende Proben zu behalten. Außerdem möchten wir dem Transporter der Proben, welcher diese von A nach B transportieren muss, ermöglichen, diese Transporte einfach protokolliert durchführen zu können, mit einfachem Probenverlustprotokoll. Zu guter Letzt soll es noch

dem System-Administrator möglich sein, die Benutzer des Systems zu verwalten und ihnen ihre Rollen im SFB zuzuteilen.

Diese Webapplikation soll also, in Stichpunkten, die folgenden Funktionen ermöglichen:

- Die Erstellung und Verwaltung von Prozessketten durch einen Prozessketten-Administrator.
- Eine protokollierte, (digital semi-automatische-) beobachtete Durchführung von Experimenten auf die Werkstoff-Proben.
- Ein protokollierter Transport der Proben in ihren Trägern von A nach B, mit Verlustprotokoll.
- Eine einfache Übersicht über die im Lager befindende Proben zu haben.
- Nutzer der Webapplikation einfach verwalten und ihren Rollen im SFB einteilen zu können.
- Das System sollte zudem auch noch lauffähig auf mobilen Geräten sein, und möchte noch auf Englisch verfügbar sein.

Leser dieser Architekturbeschreibung sind hiermit also nicht nur die Mitarbeiter der SFB-Initiative, sondern gegeben falls auch der Endkunde, welcher die Benutzer des Systems verwalten kann.

Im Folgenden wird oft an Stellen, wo natürlich User*innen gemeint sind, nur User geschrieben (und analog für beispielsweise die spezifischen Rollen der Benutzer*innen).

1.1 Zweck

Diese Architekturbeschreibung beschreibt die Architektur von Farbiges SFB, eine digitale Software welche die Erstellung, Verwaltung und Durchführung von Prozessketten ermöglicht, mit wessen Hilfe neue Werkstoffeigenschaften gefunden werden sollen für den industriellen Gebrauch. Es beschreibt, wie das System von Farbiges SFB implementiert werden soll und wie seine Struktur aussehen wird. Dieses Dokument soll allen, die die Software bedienen und bearbeiten werden, dabei helfen die Struktur und Funktionsweise von Farbiges SFB zu verstehen. Es soll sowohl den Entwicklern eine Übersicht über die Funktionsweise der Software geben, welche diese schließlich implementieren werden, als auch den Testern zur Vorbereitung ihrer Tests. Es soll außerdem dem Systemadministrator eine Übersicht der Software geben, da diese das System anschließend verwalten müssen und diese vielleicht die Software weiterentwickeln möchten.

1.2 Status

Diese Architektur beschreibt den ersten Entwurf dieser Software. Es wurde auch noch nicht freigegeben durch ein Architekturreview, und ist somit die allererste Version.

1.3 Definitionen, Akronyme und Abkürzungen

In diesem Kapitel werden Akronyme und andere Ausdrücke definiert, welche im Laufe des Dokuments verwendet werden.

Begriff	Erklärung
Das System Selbst	
Probe	Ein einzelner Werkstoff
Werkstoff	Mikrokugeln die sich durch Härte, hohe Belastbarkeit, Temperaturbeständigkeit und Abriebfestigkeit auszeichnen
Träger	Transportmittel für Proben. Arten: Einzelträger, eingebettete Träger und Glasträger. Proben werden immer in Trägern transportiert
Experimentierstation **(ES)**	Physischer Standort an dem Technologen Messungen durchführen.
Technologe	Ein Angestellter im SFB, er forscht und bedient die Experimentierstationen, und wertet diese im Anschluss aus und schreibt die Probeneigenschaften in die Software.
Prozesskette **(PK)**	Besteht aus vielen Prozessschritten, die hintereinander ohne Verzweigung ablaufen. Sobald ein Auftrag erfolgt, wird eine Prozesskette gestartet.
Prozesskettenauftrag **(PK-Auftrag)**	Enthält die Prozessparameter für jeden Schritt einer Prozesskette. Der Prozesskettenadministrator legt die Prozessparameter fest. Der Logistiker legt die Proben und Träger fest.
Prozessschritt **(PS)**	In einem Prozessschritt werden Werkstoffeigenschaften beeinflusst und / oder verändert. Er enthält Prozessparameter, welche beschreiben wie die Eigenschaften beeinflusst werden. Ein Prozessschritt kann Vorbedingungen haben und eine geschätzte Dauer.
Material- / Werkstoffeigenschaften	Jeder Werkstoff besitzt Eigenschaften. (Umformbarkeit / Verformbarkeit / ...)
Prozessparameter **(PP)**	Sind qualitativ oder quantitativ. Qualitativ beschreibt, ob etwas so ist oder nicht. Quantitative Parameter enthalten einen Namen, ein Wert und eine Einheit.

Begriff	Erklärung
Logistiker	Verwaltet das Archiv und alle Proben und Träger. Für die Ausführung einer Prozesskette bestimmt er welche Proben benutzt werden, falls dies nicht möglich ist, meldet er es dem Prozesskettenadministrator.
Prozesskettenadministrator **(PK-Admin)**	Verwaltet Prozessketten und deren Prozessschritte nach bestimmten Vorlagen. Verwaltet auch die Vorlagen. Ordnet Prozessschritten Experimentierstationen zu. Überprüft dann auf Korrektheit.
Transporter	Transportiert Träger mit ihren Proben zwischen Experimentierstationen. Bei Probenverlust meldet er diese.
Transportauftrag **(T-Auftrag)**	Enthält Start- und Zielexperimentierstationen, wird vom Transporter ausgeführt.
System Administrator **(Sys-Admin)**	Verwaltet Nutzer und Experimentierstationen. Konfiguriert globale Einstellungen und ist für Backups zuständig.
Prozessketteninstanz Werte **(PIW)**	Enthält für jeden Prozessschritt der Prozesskette alle Prozessparameter. Der Prozessketteninstanzen zu planen.
Standort	Träger und Proben können entweder an einer Experimentierstation, im Archiv oder als verloren gemeldet sein.
Transportauftragzustand **(TA-Zustand)**	Ein Transportauftrag befindet sich im Zustand wartend oder er wurde schon geliefert.
Prozessschrittzustand **(PS-Zustand)**	Ein Prozessschrittzustand hat entweder den Zustand angenommen, in Bearbeitung oder abgegeben.
Upload in Datenbank	Ein Technologe muss nach dem Ausführen eines Prozessschrittes die Prozessparameter in die Datenbank (DAVIS) hochladen.

Begriff	Erklärung
Technische Ausdrücke	
Anwendungsfälle	Hilfsmittel, um Anforderungen unseres Software-Projekts zu erfassen. Sie beschreiben also, was unser System tun soll. Hierzu nutzen wir Akteure und Use-Cases.
H2	Eine SQL Java Datenbank, die auch die JDBC API unterstützt.

Begriff	Erklärung
Apache Maven	Ist ein automatisches Buildtool für Java Projekte. Es beschreibt, wie Software gebaut wird und was ihre Dependencies sind.
DAO	Objekte, die abstrakte Interfaces zu Datenbanken oder anderen Persistenzmechanismen bieten. Kurzgesagt, werden sie verwendet, um Daten aus einer Datenbank zu verwalten (Data Access Object).
Java	Eine objektorientierte Programmiersprache, welche klassenbasiert ist. Läuft universell auf fast allen Platformen, die eine JVM haben, von PCs zu Handys usw...
JavaEE (JakartaEE)	Ist eine Menge an Spezifikationen für Java SE welche Enterprise Features und Webservices sowie Distributed Computation spezifiziert.
Hibernate Validator	
Lombok	Ist eine Java Bibliothek welche zur automatisierten Code-Erstellung verwendet weden kann (um nicht so viel Code schreiben zu müssen) z. B. automatische Erstellung von Gettern und Settern, Equals, Hashcode und String.
JUnit	Ist ein Unit Testing Framework für Java, zur Überprüfung von bekannten Fehlern in den implementierten Funktionen und Klassen.
Wildfly	Ist ein Application Server, der die JavaEE Spezifikation implementiert.
Mockito	Ist ein Mocking Framework für JUnit, zum weiteren Testen der implementierten Funktionen. Es ermöglicht die Erstellung von Test Double Objekte für automatische Unit Tests.
Scrypt	Eine passwortbasierte Schlüsselableitungsfunktion, die nicht umkehrbar ist.
Bootsfaces	Ist eine JSF Framework, welche Bootstrap und JQuery benutzt, um responsive Front-Ends zu entwickeln.
Primefaces	Ist eine Menge an UI Komponenten für JSF. Es bietet Templates und Themes an.
JSF	Steht für Java Server Faces. Ist eine Spezifikation zur Entwicklung von grafischen Oberflächen in Java.

Begriff	Erklärung
GUI	Eine Form eines User Interfaces, die dem Benutzer erlaubt durch elektronische Geräte grafisch mit der Software zu interagieren (Graphical User Interface).
Klassendiagramm	Ist ein Typ von statischen Struktur Diagrammen, welche Klassen, Attribute, Operationen und Relationen des Systems als Diagramm darstellen.
Komponentendiagramm	Beschreiben wie Komponenten durch ihre Schnittstellen verbunden sind. Sie werden benutzt um die Struktur komplexer Systeme darzustellen.
Model-View-Controller (MVC)	Ein Architekturmuster welches für User Interfaces benutzt wird um Programmlogik in 3 Elemente zu unterteilen: Model, View und Controller.
Object-Relation-Mapping (ORM)	Ist ein Verfahren um Datentypen zwischen inkompatiblen Typ Systemen zu konvertieren. Dieses erstellt eine virtuelle Objekt Datenbank
Packetdiagramm	Zeigt die Abhängigkeiten der Packete des Models.
Sequenzdiagramme	Zeigt die Struktur unseres Models und stellt den Austausch von nachricten zwischen Objekten dar. Dabei wird eine Lebenslinie benutzt.
Problemkarten	Beschreiben Probleme und passende Strategien.
SQL	Ist eine domänenspezifische Sprache, die in der Softwareentwicklung benutzt wird und Daten in relationalen Datenbanken verwaltet.

1.4 Referenzen

- Architekturbeschreibung von RainersRaiders im Rahmen von Softwareprojekt 2 - 2016
- Kickoff Folien

1.5 Übersicht über das Dokument

Im Kapitel 2 werden wir zunächst verschiedene Anwendungsfälle unserer Software beschreiben, welche wir als Ausgangspunkt für unsere Architekturstruktur verwenden werden. In Kapitel 3 werden die Einflussfaktoren, Probleme und Strategien, die zum Lösen der Probleme entworfen wurden, dargestellt. In den Kapiteln 4, 5 und 7 werden die verschiedenen Sichten von Hofmeister gezeigt. Die Codesicht wird in diesem Projekt nicht behandelt. Im nachfolgenden Kapitel 4 wird die konzeptionelle Sicht unserer Software beschrieben. In Kapitel 5 wird die Modulsicht und in Kapitel 7 wird die Ausführungssicht gezeigt. Kapitel 6 ist eine nähere Beschreibung der Architektursicht und wird durch ein Modell in der Datensicht dargestellt. Im Kapitel 8 werden einige Anwendungsfälle dargestellt. Im abschließenden Kapitel 9 wird auf die mögliche zukünftige Entwicklung der Software bei neu eingehenden Wünschen der Kunden eingegangen.

2 Anwendungsfälle

In diesem Kapitel werden wir verschiedene Anwendungsszenarien abdecken, welche unsere Software erfüllen soll. Diese Szenarien werden wir als Ausgangs- und Vergleichspunkt für den Rest unserer Architektur benutzen. Hiermit kann am Ende geprüft werden, ob die Software alle Anforderungen erfüllt.

2.1 Anmeldung am System (alle Benutzer)

Der Benutzer wird mit einem Login-Fenster begrüßt, an welchem er sich einloggen kann. Sollte es noch keinen Benutzer im System geben, so kann ein neuer Benutzer über ein Create-Account Knopf erstellt werden. Sollte er sein Passwort vergessen haben, so kann er mithilfe von einem Passwort Vergessen Knopf dieses zurücksetzen. Nach der Erstellung eines Benutzers wird dieser im Datenbanksystem eingespeichert, wodurch der Systemadministrator ihm nun eine Rolle einteilen kann. Der Systemadministrator befindet sich schon zu Anfang im System. Das heißt, er muss seinen Benutzer nicht neu erstellen. Nach einer erfolgreichen Anmeldung am System werden alle Benutzer mit einer Navigationstabelle begrüßt. Diese Tabelle beinhaltet die für den Benutzer spezifizierten Funktionalitäten.

2.2 Benutzer Verwaltung - System Administrator

Die Aufgabe des Systemadministrators ist es, die Benutzer zu verwalten. Nach seiner Anmeldung am System wird dieser mit einer Tabelle begrüßt, welche seine Optionen im System anzeigt. Nach Druck auf den Knopf für Nutzerverwaltung wird er mit einer weiteren Tabelle begrüßt, welche alle Nutzer des Systems beinhaltet. Diese Tabelle hat sowohl einen Sortierungsknopf, als auch eine Suchbox, wodurch er die angezeigten

Nutzer des Systems filtern kann. Jeder Nutzer hat einen Bearbeitungsknopf neben seinem Namen. Durch Druck auf diesen Knopf öffnet sich ein weiteres Fenster, in dem der Systemadministrator alle Optionen zur Verwaltung des Nutzers hat. Hier kann er z. B. dessen Details ändern (Name, Vorname, usw.), seine Rolle im SFB (Technologe, Transporter, usw.). Er hat aber auch einen Löschknopf, mit welchem er diesen Benutzer aus dem System entfernen kann. Die Tabelle beinhaltet natürlich auch einen Add-Knopf, wodurch der Systemadministrator neue Benutzer in das System einfügen kann.

2.3 Experimentierstationsverwaltung - System Administrator

Wir nehmen an, der Administrator hätte sich schon im System angemeldet. In dem ihm angezeigten Fenster sieht er unter der Benutzerverwaltung einen weiteren Knopf zur Experimentierstationsverwaltung. Durch Druck auf diesen Knopf öffnet sich ein weiteres Fenster mit einer weiteren Tabelle, welche alle im System enthaltenden Experimentierstationen beinhaltet. Diese sind auch wieder mal in Reihen der Tabelle eingeteilt. Jede Reihe hat rechts einen Knopf, mit welchem der Administrator die zugehörige Experimentierstation verwalten kann, und daneben einen weiteren Knopf zur Löschung der Experimentierstation. Unten in der Tabelle befindet sich ein weiterer Knopf, mit dem er weitere Experimentierstationen einfügen kann. Bei normalem Druck auf eine Experimentierstation werden Informationen über diese angezeigt, wie z. B. Standort und ob diese momentan in Benutzung ist.

2.4 Prozesskettenverwaltung - Prozesskettenadministrator

Da die Aufgabe des Prozesskettenadministrators die Erstellung und Verwaltung von Prozessketten ist, wird dieser nach seiner Anmeldung am System mit einem Fenster begrüßt, wo er durch Druck auf den Knopf zur Prozesskettenverwaltung mit einem weiteren Fenster begrüßt wird, welches alle Prozessketten des Systems enthält. Auch diese Tabelle kann durch eine Suchbox und weitere Kriterien wie z. B. Datum der Erstellung der Prozesskette filtriert werden. In dieser Tabelle sind alle Prozessketten, die sich im System befinden, in reihen sortiert sichtbar. An der Seite jeder noch nicht instanziierten Prozesskette ist ein Knopf, um diese zu bearbeiten. Durch Druck auf den Bearbeitungsknopf kann der Prozesskettenadministrator die Schritte dieser Prozesskette bearbeiten. Hier kann er z. B. weitere Schritte der Prozesskette hinzufügen, oder schon existierende Schritte entfernen. Ein weiterer Knopf in der Tabelle erlaubt ihm, eine Prozesskette zu starten. Sobald dieses getan ist, kann diese nicht mehr von ihm gestoppt werden. Die Tabelle beinhaltet auch einen Einfügungsknopf, durch welchen weitere Prozessketten erstellt werden können. Bei Erstellung einer Prozesskette wird diese auf Korrektheit überprüft. Sollte ein Prozessschritt z. B. unrealistische Vorbedingungen haben, so wird dem Prozesskettenadministrator Bescheid gegeben, dass er diese Prozesskette im jetzigen Stand nicht erstellen kann. Bei der Erstellung von Prozessketten, welche aus verschiedenen Prozessschritten besteht, muss der Prozesskettenadministrator natürlich

auch diese Prozessschritte erstellen. Bei der Erstellung von Prozessketten wird er also mit einem Fenster begrüßt, in welchem er die Prozessschritte der Prozesskette verwalten kann. Hier kann er z. B. einem Prozessschritt Vorbedingungen geben, welche er in einem Vorbedingungsfeld eingeben kann. In einem weiteren Feld kann er die Experimentierstation dieses Schrittes zuordnen. Diese Prozessschritte kann er so lange noch bearbeiten und verändern, bis die Prozesskette gestartet wird, durch den Druck auf den Knopf zur Bearbeitung der Prozesskette. Sollte eine Prozesskette in Bearbeitung sein, so kann der Prozesskettenadministrator auf diese drücken, um zu sehen, bei welchem Schritt sie sich momentan befindet. Er kann außerdem auch alle schon abgelaufenen und noch auszuführende Schritte sehen.

2.5 Lagerübersicht - Logistiker

Nehmen wir an, der Logistiker hätte sich im System schon angemeldet. In dem vor ihm befindlichen Fenster ist ein Knopf für die Lagerübersicht. Durch Druck auf diesen öffnet sich ein weiteres Fenster mit einer Tabelle, welche alle sich im Lager befindenden Proben beinhaltet. Auch diese Tabelle kann durch Suchbox und Weiteres filtriert werden. Sie beinhaltet aber auch eine Option zur Gruppierung der Proben nach Legierung, Wärmebehandlung und Stationen.

2.6 Proben mit Prozessketten assoziieren - Logistiker

Nach seinem Login findet der Logistiker in dem Begrüßungsfenster einen weiteren Knopf für Prozessketten. Hier kann er die verschiedenen Prozessketten sehen welche gerade ausgeführt werden und welche Proben sie benötigen. Durch eine schnelle Suche mit z. B. der Suchbox im Lagerübersichtsfenster kann der Logistiker herausfinden, ob die benötigten Proben sich im Lager befinden. Es soll bei jeder Prozesskette auch einen automatisierten Suchknopf geben, welcher diese Proben automatisch im Lager sucht. Befinden sich die Proben im Lager, so kann er die Proben der Prozesskette zuweisen. Sollte das nicht der Fall sein, so kann er auf einen Knopf neben der Prozesskette drücken, wodurch dem Prozesskettenadministrator Bescheid gegeben wird, dass die benötigten Proben nicht verfügbar sind.

2.7 Prozessketten Starten/Stoppen - Logistiker

Sollten die benötigten Proben vom letzten Absatz sich im Lager befinden, so kann der Logistiker durch einen weiteren Knopf eine noch nicht gestartete Prozesskette starten. Sollte, aus welchem Grund auch immer, eine Prozesskette gestoppt werden, so kann dieser auch diese stoppen.

2.8 Zustand eines Auftrags angeben - Technologe

Nehmen wir an, der Technologe hätte sich schon im System angemeldet. Er sieht vor sich ein Fenster mit einem Knopf für Aufträge. Durch Druck auf diesen gelangt er zu einem weiteren Fenster, in dem alle verfügbaren Aufträge als Reihen in einer Tabelle angezeigt werden. Hier kann er nun einen Auftrag, durch Druck auf einen Knopf neben einem Auftrag, annehmen, wodurch dieser Auftrag automatisch in den Zustand angenommen geht. Dieser Auftrag kann nun von keinem anderen Technologen mehr angenommen werden. Sobald der Technologe mit der Bearbeitung des Auftrags beginnt, muss er nur auf den Knopf für in-Bearbeitung drücken, wodurch dieser Auftrag nun automatisch in den Zustand in-Bearbeitung gelangt. Sobald der Technologe mit seiner Bearbeitung fertig ist, muss er nur auf einen Knopf drücken, wodurch der Auftrag nun als bearbeitet gekennzeichnet wird. Sollte er nun diesen Auftrag weiterschicken möchten, so kann er dies durch Druck auf einen weiteren Knopf tun. Sobald der Technologe die Prozessparameter dieses Auftrags in die Datenbank hochgeladen hat, wird der Auftrag entweder automatisch in der Datenbank auf hochgeladen gesetzt, oder manuell durch den Technologen. Dieses kommt drauf an, ob der Technologe zum Hochladen der Daten den Knopf in diesem Auftrag verwendet hat, oder einen anderen.

2.9 Aufnahme der Zeit in einem Auftrag - Technologe

Bei der Annahme eines Auftrags aus dem vorherigen Absatz wird die jetzige Systemzeit aufgenommen und im Auftrag eingegeben. Das Gleiche passiert auch bei den weiteren Schritten aus dem vorherigen Absatz. Sollte der Technologe allerdings an einem späteren Zeitpunkt diese Zeiten bearbeiten möchten, so kann er durch Druck auf einen Bearbeitungsknopf neben dem Auftrag in der Tabelle die Zeiten manuell ändern.

2.10 Station Fehler Melden - Technologe

Sollte irgendetwas schiefgehen und eine Station fehlerhaft werden, so kann der Technologe, nach seiner Anmeldung im System, durch einen Knopf einen Stationsfehler melden. Hierdurch wird das System auch die Fehlernachricht bekommen, und somit allen anderen Prozessketten, welche diese Station benötigen, darüber Bescheid geben.

2.11 Proben Transportieren - Transporteur

Nehmen wir an, der Transporter hätte sich schon am System angemeldet. Er sieht ein Fenster mit einem Transportübersicht-Knopf, welcher ein weiteres Fenster mit allen Transportaufträgen als Reihen in einer Tabelle öffnet. Hier kann er, durch Druck auf einen Auftrag sehen, wo diese Proben abgeholt werden müssen und wo sie hin geliefert werden müssen. Er kann durch Knopfdruck einen Auftrag annehmen, wodurch dieser als Annahme gekennzeichnet wird. Sobald er die Proben abgeholt hat, kann er durch

Knopfdruck diesen Auftrag auf die Kennzeichnung Transport setzen. Sobald er die Proben zum korrekten Ort gebracht hat, kann er durch einen weiteren Knopfdruck diesen Auftrag als ausgeliefert markieren.

2.12 Probenverlust Melden - Transporteur

Nachdem der Transporteur den Transport aus dem vorherigen Abschnitt ausgeführt hat, kann er durch Druck auf einen Knopf innerhalb der Auftragstabelle für jeden Auftrag Probenverlust angeben. Hierdurch wird der Auftrag automatisch auf den Zustand Verlust gesetzt.

3 Globale Analyse

Hier werden Einflussfaktoren aufgezählt und bewertet sowie Strategien zum Umgang mit interferierenden Einflussfaktoren entwickelt.

3.1 Einflussfaktoren

(Santiago)

die Notation der Flexibilität hat die Bedeutung :

- - - : Keine Flexibilität
- - : schlechte, aber nicht komplett starre Flexibilität
- ++: sehr gute Flexibilität
- +: gute Anfoderungsflexibilität

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
O1 : Organisation				
O1.1 Time-To-Market				
	Die Auslieferung erfolgt am 08.03.2020.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Nicht alle Funktionen können realisiert werden.
O1.2 Architektur-Abgabe				
	Die Auslieferung erfolgt am 22.12.2019.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Durch den Zeitdruck könnte die Architektur mangelhaft werden. Wenn wir uns nicht genug Zeit lassen, könnten Aspekte, die relevant für die Architektur sind, vergessen werden.
O1.3 Entwickler				
	Die Projektgruppe besteht aus 6 Entwicklern	Ein wenig Veränderlichkeit und Flexibilität: Gruppenmitglieder könnten die Gruppe verlassen.	o/ o	Die Architektur kann wegen Zeitmangel (es können nicht mehr als die ursprünglichen sechs Entwickler mitarbeiten) und fehlenden Fähigkeiten Mangel enthalten.
O1.4 Fähigkeiten Entwickler				
	Nicht alle Entwickler haben die gleiche Programmier-erfahrung und auch nicht mit den gleichen Technologien.	Hohe Veränderlichkeit und Flexibilität durch Ausführen des Projekts und Recherche.	++/ ++	Die Implementierung kann Mangel enthalten.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
T1: Technik				
T1.1: Programmiersprache				
	Java 11 oder höher ist vorgegeben.	Keine Flexibilität, da Teil der Mindestanforderungen. Ein wenig Veränderlichkeit an der Version der Sprache.	- -/ o	Das Projekt muss in Java umgesetzt werden.
T1.2 Webbrowser				
	Die Anwendung muss in gängigen Browsern (Firefox, Internet Explorer, Safari, Edge) laufen.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Bei der Implementierung muss darauf geachtet werden, plattformunabhängig vorzugehen.
T1.3 Server				
	Zur Implementierung muss JavaEE 8 benutzt werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Das Projekt muss komplett in Java umgesetzt werden.
T1.4: Oberfläche				
	Als Framework zur Erstellung der Oberfläche muss JSF verwendet werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Das Projekt muss in Java umgesetzt werden.
T1.5: Persistenz				
	Zur sicheren Speicherung der Daten soll die relationale Datenbank H2 verwendet werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Bei der Implementierung muss H2 verwendet werden.
T1.6: Build System				
	Maven muss als Build-System verwendet werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Das Projekt muss maven-build fähig sein.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
T1.7: DeltaSpike Data				
T1.5 P1.4	Wir verwenden DeltaSpike Data.	Große Flexibilität, dies ist keine Verpflichtung, sondern eine Entscheidung unserer Gruppe. Nach Beginn der Implementierung besteht eine bedingte Veränderlichkeit: Eventuell wären Änderungen zu aufwendig.	++/ o	Unsere Architektur muss darauf ausgelegt sein, dass DeltaSpike Data benutzt wird. Darauf muss vor allem bei den Persistenzklassen geachtet werden.
T1.8: Multiple Users				
	Die Anwendung muss von mehreren Benutzern gleichzeitig verwendbar sein.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Anwendung darf nicht von gleichzeitiger Verwendung von mehreren Nutzern überfordert sein; ebenfalls dürfen dadurch keine Sicherheitslücken entstehen.
P1: Produktfaktoren				
P1.1.1: Upload				
	Prozessketten- und Prozessschritt-Parameter sollen aus JSON-Dateien hochgeladen werden können.	Keine Veränderlichkeit, da es vom Chinese Menu ist aber Flexibilität gegeben: muss nicht unbedingt gemacht werden.	- -/ ++	Die Architektur muss vorsehen, dass JSON-Dateien hochgeladen werden können, aus denen automatisch für einen Prozessschritt/ eine Prozesskette die Parameter eingefügt werden.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.1.2: Download				
	Die Parameter von einzelnen Prozessschritten sollen nach JSON exportiert und heruntergeladen werden können.	Keine Veränderlichkeit, da es vom Chinese Menu ist aber Flexibilität gegeben: muss nicht unbedingt gemacht werden.	- -/ +	Die Architektur muss vorsehen, dass für einen Prozessschritt die Parameter gelistet und in einem JSON-Format exportiert werden können.
P1.2: Protokollierung				
P1.2.1: Protokollierung der Aufträge				
	Die Übergänge in den Aufträgen müssen protokolliert werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ -	Die Architektur muss vorsehen, dass für jeden Auftrag ein Protokoll gespeichert wird, zu dem automatisch alle notwendigen Informationen ergänzt werden.
P1.2.2: Export Protokollierung				
	Die Protokolle müssen nach JSON oder XML exportierbar sein.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss vorsehen, dass es 1. eine Möglichkeit für den Benutzer gibt, sich diese Protokolle in das Format seiner Wahl exportieren zu lassen, und 2. die Protokolle nach JSON oder XML exportierbar sind.
P1.3: Benutzerverwaltung				
	Nutzer sollen erstellt, bearbeitet und gelöscht werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Anwendung muss eine Möglichkeiten vorsehen, alle notwendigen Informationen für neue Nutzer einzugeben; des weiteren muss sie mit diesen neuen Nutzern umgehen können. Nutzer müssen auch gelöscht werden können, ohne dass es an anderen Stellen zu Informationslücken/Fehlern kommt.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.4: Experimentierstationen				
P1.4.1: Experimentierstationen verwalten				
	Stationen sollen erstellt, bearbeitet und gelöscht werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss eine Möglichkeit vorsehen, alle notwendigen Informationen für neue Stationen einzugeben. Neue Stationen müssen mit eingeplant werden (zum Beispiel bei der automatischen Auswahl, welche Experimentier-stationen ein Auftrag durchlaufen wird).
P1.4.2: Auslastung				
	Eine Übersicht über die Auslastung der Experimentier-stationen soll möglich sein.	Keine Veränderlichkeit, da es vom Chinese Menu ist aber Flexibilität gegeben: muss nicht unbedingt gemacht werden.	- -/ - -	Die Architektur muss speichern, welche Experimentierstationen belegt sind.
P1.4.3: Kaputte Stationen				
	Experimentier-stationen sollen als kaputt gemeldet werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Anwendung muss mit kaputten Stationen umgehen können: Diese sollte nicht mehr eingeplant werden. Ebenfalls sollten Experimentierstationen für Benutzer als kaputt anzeigbar sein.
P1.4.4: Warteschlange				
P1.7.1 P1.5.2	Die laufenden Aufträge, die an jeder Experimentier-station anstehen, sollen auto-matisch nach Priorität sortiert werden.	Hohe Veränderlichkeit und Flexibilität: Dies ist keine Anforderung. Wir können uns auch nach Beginn der Implementierung noch ohne größere Probleme dazu entscheiden, diese Verantwortung dem User zu übertragen.	++/ ++	Die Anwendung muss die Prioritäten analysieren können, um herauszufinden, in welcher Reihenfolge die Aufträge abgearbeitet werden sollen.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.4.4: Warteschlange				
P1.7.1 P1.5.2	Die laufenden Aufträge, die an jeder Experimentierstation anstehen, sollen automatisch nach Priorität sortiert werden.	Hohe Veränderlichkeit und Flexibilität: Dies ist keine Anforderung, sondern eine Entscheidung der Gruppe. Wir können uns auch nach Beginn der Implementierung noch ohne größere Probleme dazu entscheiden, diese Verantwortung dem User zu übertragen.	++/ ++	Die Anwendung muss die Prioritäten analysieren können, um herauszufinden, in welcher Reihenfolge die Aufträge abgearbeitet werden sollen.
P1.5: Prozessschritte				
P1.5.1: Grundlagen Prozessschritte				
	Prozessschritte sollen erstellt, gelöscht, bearbeitet, und hervorgehoben (sofern im Zustand kaputt) werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss eine Möglichkeit vorsehen, alle notwendigen Informationen für neue Prozessschritte einzugeben. Des Weiteren muss die Architektur automatisch erkennen, wenn Schritte im Zustand auf kaputt gesetzt sind.
P1.5.2: Prozessschritte an Experimentierstationen				
	Für jeden Prozessschritt soll gespeichert werden, an welchen Stationen dieser durchgeführt werden kann.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss diese Information speichern und damit umgehen können, dass es eventuell im Ablauf einer Prozesskette mehrere Optionen gibt.
P1.6: Prozessketten				
P1.6.1: Prozessketten Grundlagen				
	Prozessketten sollen erstellt, gelöscht, und bearbeitet werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss eine Möglichkeit vorsehen, die notwendigen Informationen für neue Prozessketten einzugeben.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.6.2: Prozesskettenart				
	Jede Prozesskette ist einem Typ zugeordnet.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die unterschiedlichen Arten müssen irgendwie gespeichert werden.
P1.6.3: Zustand Prozessketten				
	Für jede Prozesskette soll angegeben werden können, welche Zustände sie durchlaufen kann.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss es möglich machen, dass in den Zuständen navigiert wird (von einem zum nächsten).
P1.7: Aufträge				
P1.7.1: Aufträge Grundlagen				
	Aufträge sollen erstellt, freigegeben, gestoppt, gelöscht, priorisiert, und bearbeitet werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss mit priorisierten Aufträgen umgehen können. Des weiteren muss es möglich sein, aus Prozessketten Aufträge zu erstellen.
P1.7.2: Zuordnen zu Aufträgen				
	Aufträgen sollen Proben/Trägern zugeordnet werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss vorsehen, dass Proben und Träger irgendwie Aufträgen zugeordnet werden können. Zugeordnete Proben/Träger dürfen dann nicht mehr anderen Aufträgen zugeordnet werden; was von der Anwendung überprüft werden muss.
P1.7.3: Automatisches Zuordnen				
	Alternativ sollen Aufträgen automatisch Proben/Trägern zugeordnet werden.	Keine Veränderlichkeit, aber Flexibilität: da vom Chinese Menu, ist diese Anforderung optional.	- -/ ++	Die Anwendung muss die Parameter der Prozessschritte einer Prozesskette auswerten können, und daraus schließen, welche Proben/Träger gebraucht werden.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.7.4: Zuordnen zu Stationen				
P1.4.2 P1.5.2	Beim Starten eines Auftrags soll automatisch ermittelt werden, welche Experimentierstationen durchlaufen werden (wenn für einen Schritt mehrere möglich sind).	Hohe Veränderlichkeit und Flexibilität: Dies ist keine Anforderung, sondern eine Entscheidung unserer Gruppe. Auch nach Beginn der Implementierung können wir uns noch spontan dafür entscheiden, diese Verantwortung dem User zu übertragen.	++/ ++	Die Anwendung muss automatisch die Auslastung von Experimentierstationen auswerten können und die effizienteste Wahl treffen.
P1.7.5: Zustand Prozessschritt				
	Der Zustand eines Auftrags soll manuell aktualisiert werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss vorsehen, dass von außen eine Auwahl des Zustandes möglich ist.
P1.8: Träger				
P1.8.1: Träger Grundlagen				
	Träger sollen erstellt und gelöscht werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss eine Möglichkeit haben, Träger zu löschen (über einen Knopf oä) und neue Träger zu erstellen.
P1.8.2: Trägerarten				
	Für jeden Träger soll gespeichert werden, von welcher Art er ist.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Träger müssen ein Attribut Art haben.
P1.9: Dynamische Zustandsautomaten				
	Lineare Zustandsautomaten für die Prozessschritte sollen angelegt, gelöscht und bearbeitet werden können.	Keine Veränderlichkeit, aber Flexibilität: da vom Chinese Menu, ist diese Anforderung optional.	- -/ ++	Es muss eine Ansicht geben, in der diese Automaten erstellt werden/eine Übersicht über existierende möglich ist.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.10: Proben				
P1.10.1: Kaputte Proben				
	Proben sollen als kaputt gemeldet werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss vorsehen, dass eine Probe unterschiedliche Zustände hat (kaputt/nicht kaputt) und dass der Übergang vom Nutzer hervorgerufen wird.
P1.10.2: Verlorene Proben				
	Proben sollen als verloren gemeldet werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss mit verlorenen Proben umgehen können: Es muss möglich sein, zu sehen, wenn eine Probe verloren gegangen ist, diese darf nicht weiter verplant werden.
P1.10.3: Archivierte Proben				
	Proben müssen archiviert werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss vorsehen, dass Proben archiviert werden, wobei die zustäzlichen notwendigen Informationen gespeichert werden sollen und die Probe im Lager landet, wo sie kein Teil von einer laufenden Prozesskette ist.
P1.10.4: Lagerübersicht				
	Proben müssen archiviert werden können. Die im Lager liegenden Proben sollen angezeigt werden können.	Keine Veränderlichkeit, da es vom Chinese Menu ist aber Flexibilität gegeben: muss nicht unbedingt gemacht werden.	- -/ ++	Es muss irgendwie erkennbar sein, welche Proben im Lager sind, und welche nicht. Weitergehend muss es einfach sein,alle im Lager liegenden zu finden und aufzulisten.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1.10.5: Proben-id-Nummern				
P1.10.2	Die Probenkügelchen sollen alle eigene id-Nummern bekommen.	Flexibilität: nicht hoch, da eine Entscheidung bezüglich der Speicherung der Proben gemacht werden muss. (Allerdings auch andere Optionen theoretisch möglich). Veränderlichkeit: nach Beginn der Implementierung sehr gering, da großer Einfluss auf viele Programmteile.	-/- -	Es ensteht eine hohe Menge an Einträgen in der Datenbank, mit der ohne Performanceprobleme umgegangen werden muss.
P1.10.6:Proben Kommentar				
	Zu Proben sollen Kommentare erstellt werden können.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Es muss eine Fläche für Kommentareingabe und Darstellung geben.
P1.11: Transporte				
	Transporteure sollen Transportaufträge annehmen können. Sie sollen Proben zwischen den einzelnen Stationen tragen.	Keine Veränderlichkeit, da es vom Chinese Menu ist aber Flexibilität gegeben: muss nicht unbedingt gemacht werden	- -/ +	Die Architektur muss Transportaufträge sinnvoll in die Prozesskette einbauen können, sodass die Proben zwischen den Prozessschritten transportiert werden und an den richtigen Stellen sind.
P2: Benutzerschnittstelle				
P2.1:Mehrsprachig				
	Der User soll zwischen Deutsch und Englisch entscheiden können.	Keine Veränderlichkeit, da es vom Chinese Menu ist aber Flexibilität gegeben: muss nicht unbedingt gemacht werden.	- -/ ++	Die Architektur muss ein Umschalten zwischen den beiden Sprachen in der Darstellung vorsehen.

Abgeleitet aus	Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P2.2: Anzeige				
	Je nach Rechten des Benutzers sollen Prozessschritte, -ketten, Stationen, Proben,Aufträge, Benutzer, dynamische Abläufe und Transportaufträge angezeigt werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss eine Vielzahl an Informationen aus der Datenbank auslesen und anzeigen können, mit dem Hinblick darauf, dass es sich teilweise um sehr große Mengen handelt (Proben).
P3: Verlässlichkeit				
P3.1: User Rechte				
	Es soll unterschiedliche Rollen mit unterschiedlichen Rechten und angezeigten Informationen geben. Jeder User soll mindestens eine Rolle haben.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss zwischen diesen Rollen unterscheiden können und sichergehen, dass bestimmte Funktionen und Informationen nur zur Verfügung stehen wenn der Benutzer die Rechte dafür hat.
P3.2: Authentifizierung				
	Benutzer sollten sich in das System einloggen müssen, um relevante Informationen angezeigt zu bekommen.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss eine Login Seite vorsehen; ein Zugriff auf die Anwendung ohne sich einzuloggen soll nicht möglich sein. Deswegen muss die Architektur auch eine Sicherheitskomponente enthalten.

3.2 Probleme und Strategien

Im Folgenden werden einige Probleme, die sich aus den herausgearbeiteten Einflussfaktoren ergeben, thematisiert. Dazu werden die Probleme zunächst mitsamt der Einflussfaktoren, aus denen sie sich ergeben, vorgestellt, und anschließend unterschiedliche Strategien, um mit ihnen umzugehen, vorgestellt.
Diese Strategien werden daraufhin bewertet und es wird eine für jedes Problem ausgewählt. Diese Auswahl wird begründet.
Folgende Strategien zur Lösung von Problemen wurden ausgewählt:

- Enumeration
- Apache Shiro
- Eigene Klassen
- jakartaEE
- Datenklassen
- DeltaSpikeData
- Automatische Sortierung
- Alle einzeln
- Priorisierung Anforderungen
- Gruppenarbeit
- Eigenschaften
- Modularisierung
- Lazy Loading
- Klasse mit Zuständen
- Automatisch
- Kapselung der Datenhaltung
- Archiv-Attribut
- Als eigene Klasse
- Als Enum
- Als eigene Klasse
- als String
- Als Tupel

Problem 1: Userrollen
Es soll unterschiedliche Rollen geben, die unterschiedliche Rechte/Fähigkeiten haben; jeder User kann mehrere Rollen (muss mindestens eine) übernehmen.
Einflussfaktoren: T 1.5: Persistenz P3.1: User Rechte O 1.1: Time-To-Market
Strategien
Strategie 1.1: Mehrere Accounts: Jeder Mitarbeiter kriegt für jede der Rollen, die er erfüllt, einen eigenen Account. **Strategie 1.2: Enumeration (augewählt)** Die Rollen werden als Aufzählungstyp realisiert, der in einer m:m Beziehung zu der Userklasse steht. Dadurch kann für jeden User gespeichert werden, welche Rollen er/sie innehat. Die Sicherheit bezüglich der unterschiedlichen Fähigkeiten/Rechte der Rollen wird über Apache Shiro realisiert (siehe Strategie 1 für Problem 2)

Tabelle 3: Userrollen Problemkarte

Hier haben wir uns für die zweite Strategie entschieden, weil diese eine elegantere Lösung bietet, und benutzerfreundlicher ist. Des Weiteren erleichtert uns die bereits implementierte Java-Spezifikation die Arbeit.

Problem 2: Datensicherheit
Jeder User darf nur die Informationen sehen, die ihm nach seiner Rolle/seinen Rollen zustehen. Unauthorisierten Personen sollen keine Informationen angezeigt werden. Auch die Bearbeitung von Informationen ist nur Rollen mit den entsprechenden Privilegien erlaubt.
Einflussfaktoren: T 1.5: Persistenz P2.2:Anzeige P3.1: User Rechte P3.2: Authentifizierung
Strategien
Strategie 2.1: Apache Shiro: (ausgewählt) Das System wird Apache Shiro nutzen, um das Rollensystem zu realisieren und die Informationen auf der Datenbank vor unautorisierten Personen zu schützen. **Strategie 2.2: Eigene Implementierung:** Wir implementieren selber ein System zur Authentifizierung von Usern.

Tabelle 4: Datensicherheit Problemkarte

Hier haben wir uns für die erste Strategie entschieden, weil eine eigene Implementierung uns wahrscheinlich viel Zeit kosten würde, die wir an dieser Stelle nicht investieren müssen.

Problem 3: Vorlagen und Instanzen
Der Prozesskettenadministrator möchte nicht gestartete Prozessketten und Prozessschritte bearbeiten können.
Einflussfaktoren: P1.6.1: Prozessketten Grundlagen P1.5: Prozessschritte
Strategien
Strategie 3.1: Eigene Klassen: (ausgewählt) Wir differenzieren zwischen Vorlagen und tatsächlich gestarteten Prozessketten/schritten, beides bekommt eine eigene Klasse.

Tabelle 5: Vorlagen und Instanzen

Hier haben wir uns für die erste Strategie entschieden, weil es die einzige Möglichkeit ist, die wir gefunden haben, mit der wir die unterschiedlichen Assoziationen, die die

Vorlagen und Instanzen haben sowie die dazukommenden Parameter der Instanzen von Schritten und Ketten darstellen können.

Problem 4: Upload/Download Prozessketten- und Prozessschrittparameter
Die Software kann neue Prozessketten- und Prozessschrittparameter aus JSON-Dateien hochladen, und bestehende als JSON-Dateien exportieren.
Einflussfaktoren: T 1.5: Persistenz P 1.1.1: Upload P 1.1.2: Download
Strategien
Strategie 4.1: JAXB: Konkrete Parameter sind bei uns Objekte von einer Klasse Parameter. Diese Objekte können wir mithilfe von JAXB und der Rest API nach JSON serialisieren. **Strategie 4.2: jakartaEE:(ausgewählt)** Wir benutzen JSON-B mit jakarteEE, um die Parameter zu serialisieren.

Tabelle 6: Upload/Download Problemkarte

Hier haben wir uns für die zweite Strategie entschieden, weil diese Umsetzung weniger aufwändig zu sein scheint. Des Weiteren sollen wir sowieso schon JakartaEE benutzen.

Problem 5: Protokollierung der Prozessketten
Die Abarbeitung einer Prozesskette soll protokolliert werden (wann welche Zustandsübergänge stattfanden). Dazu sollen Parameter, Zeitpunkte und Stationen gespeichert werden.
Einflussfaktoren: T 1.5: Persistenz P 1.2.1: Protokollierung der Prozessketten
Strategien
Strategie 5.1: Datenklassen (ausgewählt) Wir legen für die Protokolle eigene Datenklassen an. **Strategie 5.2: Liste von Strings** Alle Details werden in einer Liste von Strings, die jeweils den Protokolleintrag für eine Veränderung beinhalten, gespeichert.

Tabelle 7: Protokollierung der Prozessketten Problem Karte

Hier haben wir uns für die erste Strategie entschieden, weil diese eine feste Formatierung vorgibt, und somit sichergestellt ist, dass die Protokolle gleich aussehen.

Problem 6: Stationenbearbeitung
Der Administrator soll neue Stationen hinzufügen können. Die Software muss die Fähigkeit haben neue Stationen zu übernehmen und zu nutzen. Außerdem soll der Zustand jeder Station sichtbar und bearbeitbar sein.
Einflussfaktoren: T 1.5: Persistenz P 1.4.1: Experimentierstationen verwalten
Strategien
Strategie 6.1: DeltaSpike Data: (ausgewählt) Wir benutzen DeltaSpike Data, um Repository Patterns zu implementieren, zum Beispiel für die Bearbeitung von Stationen.

Tabelle 8: Startionenbearbeitung Problemkarte

Hier haben wir uns für die erste Strategie entschieden, da sich dadurch eine praktische Abkoppelung zwischen Datenbank-Queries und Buisnesslogik ergibt. Des Weiteren sparen wir uns durch die Verwendung Zeit.

Problem 7: Priorisieren von Prozessketten
Laufende Prozessketten sollen priorisiert werden können, was zur Folge hat, dass sie, wann immer mehrere Prozessketten auf die gleichen Aktionen warten, schneller drankommen.
Einflussfaktoren: P 1.7.1: Aufträge Grundlagen P1.5.2: Prozessschritte an Experimentierstationen
Strategien
Strategie 7.1: Anzeige Die Prioritäten der Prozessketten werden den Usern angezeigt, damit sie selber sehen, welche sie als Erstes bearbeiten sollten. **Strategie 7.2: Automatische Sortierung: (ausgewählt)** Es werden automatisch die Prozesskettenprioritäten analysiert und sortiert angezeigt.

Tabelle 9: Priorisieren von Prozessketten Problemkarte

Hier haben wir uns für die zweite Strategie entschieden, weil sie deutlich benutzer-

freundlicher ist.

Problem 8: Probenverwaltung
In der Anwendungsumgebung gibt es eine hohe Anzahl an Proben, die alle in der Datenbank gespeichert werden sollen. Dadurch, dass es so viele sind (von Materialien der gleichen Sorte gibt es viele kleine Kügelchen), stellt sich die Frage, ob es sinnvoll ist, jede Einzelne dieser Kügelchen identifizieren zu können.
Einflussfaktoren: P1.10.4: Lagerübersicht
Strategien
Strategie 8.1: Alle einzeln (ausgewählt): Jedem einzelnen Probenkügelchen wird eine eigene ID zugeordnet. **Strategie 8.2: In Gruppen** Gleichartigen Probenkügelchen wird eine ID zugeordnet.

Tabelle 10: Probenverwaltung Problemkarte

Hier haben wir uns für die erste Strategie entschieden, weil dadurch genaueres Zugreifen auf die Proben möglich ist. Des Weiteren würde die zweite Strategie möglicherweise Probleme aufwerfen, wenn an einer Stelle im Lauf einer Prozesskette die Proben aufgeteilt werden und unterschiedlich weiterbehandelt werden.

Problem 9: Ambitionierter Zeitplan
Die Auslieferung der Anwendung muss am 08.03.2020 erfolgen. Dadurch könnten die Entwickler unter Zeitdruck geraten und eventuell nicht alle Anforderungen erfüllen.
Einflussfaktoren: O1.1: Time-To-Market Alle Produktfaktoren
Strategien
Strategie 9.1: Priorisierung Anforderungen:(ausgewählt) Bestimmte, notwendigere Anforderungen, sollen über andere priorisiert werden. Die höherpriorisierten werden zuerst abgearbeitet. **Strategie 9.2: Wegfallen Anforderungen:** Einige Anforderungen sind für ein Bestehen nicht notwendig und können weggelassen werden. **Strategie 9.3: Gruppenarbeit: (ausgewählt)** Die Gruppe muss strukturiert und mit möglichst fair verteilten Aufgaben arbeiten, um die Fähigkeiten aller möglichst effizient nutzen zu können.

Tabelle 11: Ambitionierter Zeitplan Problemkarte

Hier haben wir uns für die erste und dritte Strategie entschieden, weil wir grundsätzlich so viele Anforderungen wie möglich umsetzen wollen. Durch die Priorisierung ist es möglich, uns die Anzahl an Anforderungen offen zu halten, gleichzeitig aber auch ein logisches Arbeitsvorgehen zu haben. Strategie 3 ist offensichtlich.

Problem 10: Probenzuordnung zu Aufträgen
Proben sollen ohne Einfluss des Benutzers Aufträgen passend zugeordnet werden. Das bedeutet, dass das System die Eigenschaften von Proben kennen und den Anforderungen von Aufträgen passend zuordnen soll.
Einflussfaktoren: P1.7.3: Automatisches Zuordnen
Strategien
Strategie 10.1:Wegfallen Anforderungen: Diese Anforderung ist optional, das heißt, zur Not kann man ihn einfach weglassen. **Strategie 10.2: Eigenschaften: (ausgewählt)** Die Eigenschaften/Anforderungen von Proben/Aufträgen sollten nicht einfach nur String-Felder sein, da diese miteinander vergleichen schwierig ist (unterschiedliche Formulierungen etc). Stattdessen sollten Eigenschaften/Anforderungen vordefiniert sein (vom Prozesskettenadministrator?) und gespeichert werden, welche Eigenschaften zu welchen Anforderungen passen.

Tabelle 12: Probenzuordnung zu Aufträgen Problemkarte

Hier haben wir uns für Strategie 2 entschieden, weil wir nicht von Anfang an Anforderungen schon ausschließen wollen.

Problem 11: Kompetenz der Entwickler
Die unterschiedlichen Entwickler haben unterschiedlich (weitreichende) Kenntnisse der Technologien; das führt dazu, dass die Implementierung eventuell länger dauert/fehlerbehaftet ist. Aneignen dieser Informationen ist vor allem wegen dem ambitionierten Zeitplan stressig.
Einflussfaktoren: O1.1: Time-To-Market O1.3: Entwickler O1.4: Fähigkeiten Entwickler T1.1: Programmiersprache T1.2: Webbrowser T1.3: Server T1.4: Oberfläche T1.5: Persistenz T1.6: Build System
Strategien
Strategie 11.1: Kleingruppen: Die Entwickler könnten nur in Kleingruppen programmieren. Dadurch sitzen immer eine Menge unterschiedlicher Menschen mit eventuell unterschiedlichen Fähigkeiten an einem Programmstück. **Strategie 11.2: Modularisierung: (ausgewählt)** Die Implementierung wird bestmöglich nach dem Können der Entwickler aufgeteilt, indem die Anwendung modularisiert wird. Dadurch kann jeder Entwickler an seinem eigenen Modul arbeiten, und sich auf einen kleineren Bereich konzentrieren. Die Entwickler bemühen sich frühzeitig darum, sich weiteres Wissen anzueignen. Ebenfalls unterstützen sich alle Gruppenmitglieder gegenseitig.

Tabelle 13: Kompetenz der Entwickler Problemkarte

Hier haben wir uns für Strategie 2 entschieden, weil durch die Modularisierung kein Entwickler alle neuen Technologien/Themen verstehen muss, sondern jeder nur einen kleineren Teil des Ganzen. Des Weiteren ist diese Strategie hoffentlich zeiteffizienter.

Problem 12: Laden von Informationen
Der Logistiker möchte einen Überblick über alle Proben, die im Lager liegen. Da die Probenkügelchen alle einzeln gespeichert werden, und es somit eine sehr große Menge an Proben ist, könnte das Laden aller Proben auf einmal zu Performanceproblemen führen.
Einflussfaktoren: P1.10.4: Lagerübersicht P1.10.5: Proben-id-Nummern T1.5: Persistent P2.2: Anzeige
Strategien **Strategie 12.1: Lazy Loading: (ausgewählt)** Es wird immer nur ein kleiner Ausschnitt aus der Datenbank geladen (lazy loading). **Strategie 12.2: Repräsentanten:** Es werden Repräsentanten von Probenkügelchenmenge ausgewählt und angezeigt, dazu noch eine Gesamtanzahl der gleichartigen Kügelchen.

Tabelle 14: Laden von Informationen Problemkarte

Hier haben wir uns für die erste Strategie entschieden, weil diese einfacher umzusetzen ist, wodurch wir Zeit sparen, die wir auf wichtigere Aspekte der Anwendung verwenden können.

Problem 13: Nur ein Transporteur
Wenn es Träger gibt, die von einer Station zur nächsten transportiert werden sollen, dann werden diese für alle Transporteure angezeigt. Natürlich kann aber nur ein Transporteur tatsächlich die Träger weitertragen; deswegen soll es möglich sein, dass ein Transporteur einen Auftrag annimmt, der dann für alle anderen nicht mehr sichtbar ist.
Einflussfaktoren: P1.11: Transporte
Strategien **Strategie 13.1: Klasse mit Zuständen: (ausgewählt)** Es soll eine Klasse für die Transportaufträge geben. Der Transportauftrag kann unterschiedliche Zustände annehmen, es werden nur Transportaufträge dargestellt, die im entsprechenden Zustand sind. **Strategie 13.2: Attribut:** Jeder Transportauftrag hat ein Boolean Attribut, das angibt, ob er schon abgeholt worden ist.

Tabelle 15: Nur ein Transporteur Problemkarte

Hier haben wir uns für die erste Strategie entschieden, weil wir sowieso schon einen Zustandsautomaten für den Transporteur geben soll. Es ist also kein großer Aufwand, dort noch einen Zustand hinzuzufügen. Des Weiteren ist nicht zu erwarten, dass neue Zustände hinzukommen: Aufträge sind entweder noch frei, auf dem Weg, oder abgeliefert.

Problem 14: Automatische Lastverteilung
Wenn für einen Prozessschritt mehrere Experimentierstationen möglich sind, muss ausgewählt werden, welche Stationen eine Prozesskette tatsächlich benutzen soll.
Einflussfaktoren: P1.4.2: Auslastung P1.5.2: Prozessschritte an Experimentierstationen
Strategien **Strategie 14.1: Von Hand:** Der Prozesskettenadministrator wählt die Stationen selber aus, wenn er die Prozesskette instanziiert. **Strategie 14.2: Automatisch: (ausgewählt)** Das System analysiert, wie die für jeden Prozessschritt möglichen Stationen ausgelastet sind, und entscheidet, wo welche Schritte am effizientesten abgearbeitet werden.

Tabelle 16: Automatische Lastverteilung Problemkarte

Hier haben wir uns für die zweite Strategie entschieden, da sie den Ablauf im Fachbereich effizienter gestalten kann. Des Weiteren könnte es für den Prozesskettenadministrator bei zu vielen Prozessketten schwierig werden, einen Überblick zu behalten.

Problem 15: Datenbearbeitung und Datenüberwachung
Die Nutzer können mit den Daten des Systems interagieren (durch unterschiedliches, Vorausgesetzes Verhalten). Die Daten sollen sichtbar, speicherbar und bearbeitbar sein. Daher braucht das System eine geordnete Weise jede verlangte Funktionalität zu implementieren.
Einflussfaktoren: T1.7: Multiple Users T1.5: Persistenz
Strategien
Strategie 15.1: Kapselung des Datenhaltung: (ausgewählt) Nutzung unterschiedlichen Komponenten, die unterschiedlichen abstrahiere Teilen der System trägen können, damit eine schneller und sicherer Implementierung zu erreichen. (Santiago)

Tabelle 17: Prozessbearbeitung und Überwachung Problemkarte

Hier haben wir uns für die erste Strategie entschieden, weil durch die Kapselung eine Ordnung und Struktur in das System kommt.

Problem 16: Archivierung
Proben sollen archiviert werden können, dafür soll zusätzlich zu den anderen Informationen auch gespeichert werden, wann die Archivierung stattfand, und im Kontext von welchem Auftrag.
Einflussfaktoren: P1.10.4: Lagerübersicht P1.10.3: Archivierte Proben
Strategien
Strategie 16.1:Archiv-Attribut(ausgewählt) Wenn eine Probe archiviert wird, wird zusätzlich ein Attribut Archiv gesetzt, in dem die zusätzlichen Informationen gespeichert werden.

Tabelle 18: Archivierung

Hier haben wir uns für die erste Strategie entschieden, weil durch eine eigene Klasse und ein Attribut von dieser Klasse die Informationen am Besten an einem Ort gespeichert werden können.

Problem 17: Träger und Trägerart
Jeder Träger soll eine eigene Trägerart haben. Hierbei ist die Frage, wie diese Arten realisiert werden sollen.
Einflussfaktoren: P1.8: Träger
Strategien
Strategie 17.1: als String: Die Trägerart wird als Stringattribut gespeichert. **Strategie 17.2: als Enum:** Die Trägerart wird als Enum bereitgestellt. **Strategie 17.3: als eigene Klasse: (ausgewählt)** Die Trägerart wird als eigene Klasse (möglicherweise mit Attributen) dargestellt.

Tabelle 19: Träger und Trägerart Problemkarte

Hier haben wir uns für die dritte Strategie entschieden. Die Darstellung als Enum könnte Probleme aufwerfen, wenn in während der Laufzeit neue Trägerarten hinzugefügt

werden sollen: Das ginge nicht. Dafür müsste die Anwendung komplett neu übersetzt und deployed werden. Die Darstellung als String ist nicht sehr elegant und sicher. Durch die Auswahl aus Objekten der Klasse Trägerart werden mögliche Fehler eingedämmt.

Problem 18: Zustandsautomaten Prozessketten
Für die Prozessketten soll es Zustandsautomaten geben.
Einflussfaktoren: P1.6.1: Prozessketten Grundlagen P1.6.3: Zustand Prozesskette
Strategien
Strategie 18.1: als Enum: (ausgewählt) Da die Zustände fest vorgegeben sind und die Anzahl nicht veränderbar sein sollen, können sie als Enum gespeichert werden. **Strategie 18.2: als String:** Die Zustände werden als String-Liste gespeichert.

Tabelle 20: Zustandsautomaten Prozessketten Problemkarte

Hier haben wir uns für die erste Strategie entschieden. Weil die Anzahl der Zustände fest ist und nicht von irgendeinem User verändert werden soll, ergeben sich dadurch keine Probleme, und machen die Implementierung sicherer vor unerlaubten Manipulationen.

Problem 19: Zustandsautomaten Prozessschritte
Für die Prozessschritte soll es Zustandsautomaten geben.
Einflussfaktoren: P1.9: Dynamische Zustandsautomaten
Strategien
Strategie 19.1: als Enum: Die Zustände werden als Enum gespeichert. **Strategie 19.2: als Liste von Strings:** Die Zustände werden als String-Liste gespeichert. **Strategie 19.3: als eigene Klasse (ausgewählt)** Die Zustände werden als eigene Klasse gespeichert.

Tabelle 21: Zustandsautomaten Prozessschritte Problemkarte

Hier haben wir uns für die dritte Strategie entschieden, weil die Zustände wegen einer optionalen Anforderung vom Prozesskettenadministrator veränderbar sein sollen. Dafür

bietet sich die Darstellung als Enum nicht an.

Problem 20: Arten an Prozessketten
Die Prozessketten werden grundsätzlich in drei unterschiedliche Arten unterteilt; für jede Prozesskettenvorlage soll gespeichert werden, um welche es sich handelt.
Einflussfaktoren: P1.6.1: Prozessketten Grundlagen P1.6.2: Prozesskettenart
Strategien
Strategie 20.1: als Enum: (ausgewählt) Die unterschiedlichen Arten werden als Enum dargestellt. **Strategie 20.2: als String:** Für jede Vorlage wird in einem Stringattribut gespeichert, um welche Art es sich handelt.

Tabelle 22: Arten an Prozessketten Problemkarte

Hier haben wir uns für die erste Strategie entschieden, weil die Anzahl an Arten voraussichtlich nicht verändert wird. Dadurch ist nicht zu erwarten, dass der hohe Aufwand, dem Enum eine Art hinzuzufügen, aufgewandt werden muss.

Problem 21: Kommentare
Für Proben sollen Kommentare erstellt werden können.
Einflussfaktoren: P1.10.6: Proben Kommentar
Strategien
Strategie 21.1: als eigene Klasse: Es gibt eine Klasse Kommentare; in Objekten von dieser werden die Kommentare und Details gespeichert. **Strategie 21.2: Als String:** Die Kommentare zu Proben werden als Strings gespeichert. **Strategie 21.3: Als Tupel: (ausgewählt)** Die Kommentare werden als Tupel aus dem Kommentar und dem Zeitstempel gespeichert.

Tabelle 23: Kommentare Problemkarte

Hier haben wir uns für die dritte Strategie entschieden, weil wir so ohne den Aufwand

einer eigenen Klasse die nötigen Daten speichern können.

4 Konzeptionelle Sicht

In diesem Kapitel verwenden wir die Konzeptionelle Sicht nach Hofmeister et al. um das zu entwerfende System zu beschreiben. Unser System ist eine Webapplikation, die auf JSF basiert. Benutzer können mit ihren Endgeräten auf unser System zugreifen. Dafür stellen wir den Benutzern ein Web-Frontend zur Verfügung. Die Kommunikation vom JSF, unserem Framework, ist als Model-View-Controller System konzipiert worden. Dieses spiegelt sich dadurch wieder, dass der Benutzer nur auf die GUI zugreift, welche im Hintergrund einen Controller verwendet, der die Logik im System ausführt.

H2: Die Datenbanksoftware H2 wird genutzt um unser Modul zu persistieren. Sie wird benutzt, da es in den Anforderungen von uns gefordert wird. Unsere Datenbank kommuniziert über JDBC. Zudem nutzen wir DeltaSpike data von Apache. Dies ist eine Menge von CDI-Erweiterungen, die es uns ermöglicht, auf Entitäten und Klassen mit unserer JPA zugreifen zu können.

Persistence: Hier liegt die Verbindung zwischen dem System und der Datenbank. Die Persistenz ist dafür verantwortlich, dass die Datenbank auf dem akutellen Stand des laufendem Systems ist. Falls unser System terminiert, müssen alle Daten persistiert bleiben und auch bei der nächsten Benutzung noch gespeichert sein. JPA ist eine Java Spezifikation, welche unter anderem von Hibernate umgesetzt wird. Unsere Konzeption sieht es vor, dass Hibernate impliziert die JPA mittels Deltaspike data umsetzt. Die Persistenz ist mit der BusinessLogic verbunden.

BusinessLogic: Die Businesslogic beinhaltet die Funktionalität des Systems. Sie nutzt die Schnittstellen Data von Persistence und nutzt die Updateschnittstelle von Model. Sie ergibt sich aus Strategie 15.1 (3.2). Wir nutzen die Businesslogic, um die Wünsche unseres Kunden zu erfüllen. Die Businesslogic erlaubt es, dass Prozessketten geplant und durchgeführt werden können. Hier liegt also die eigentliche Funktionalität unseres Systems. Alle Controller und Beans die Geschäftsrelevante Prozesse bearbeiten und mit diesen interagieren, werden von uns mit Businesslogic benannt.

Hierzu sind die Controller mit unserer GUI verknüpft. Sollte der User irgendeine Auswahl oder Eingabe in der GUI machen, so werden die Daten direkt an den Controller übermittelt, welcher dann die Logik ausführt, indem er mit den Datenklassen und ihren Datenbankklassen(DAO) verbunden ist, und somit vollen Zugriff auf die funktionalen Schnittstellen des Systems hat.

Model: Das Model unserer Datenbank beschreibt die logische Struktur unserer Datenbank, hiermit wird fundamental bestimmt, in welche Art auf die in der Datenbank enthaltenen Daten zugegriffen werden kann. Wir nutzen eine relationale Datenbank, welche auf Tabellen basiert.

In dem Modell liegen die Daten der User, Prozessketten, Proben sowie Daten und Vormerkungen des Systems. Model stellt ein Interface Update zur Verfügung, welches von Businesslogic und Persistence genutzt wird. Diese Komponente ergibt sich aus der Strategie 15.1 (3.2).
In dem Abschnitt 6 nehmen wir unser Datenmodel genauer unter die Lupe und gehen auf die Strategien ein, welche wir versucht haben im Datenmodel umzusetzen.

Interface: Hier liegt unser User Interface, also die xhtml Website, in der die User mit dem System interagieren. Interface nutzt das Interface Lieferung von Controller. Die Komponente Interface bezieht sich auf Strategie 1.2 (3.2), weil jeder User eine unterschiedliche Oberfläche benutzt. Außerdem ergibt sich die Komponente aus der Strategie 15.1 (3.2). Interface ist mit einem Port verbunden.

Controller: Controller nutzt das Interface Control von Businesslogic und Control von Model. Controller stellt ein Interface Lieferung zur Verfügung. Durch Controller kann das Interface die unterschiedlichen Funktionen aus Businesslogic oder Model erreichen. Es ergibt sich aus der Strategie 15.1 (3.2).

REST: Hier greift der REST-Client auf REST-Funktionen zu. Über einen Port verbindet er sich mit dem REST-Client.

RESTClient: Der REST-Client ist für die Kommunikation in einem System, das auf mehreren Systemen verteilt läuft.

User: Der User greift über den Webbrowser über HTTPS/Request/Response auf das vom Interface und dessen Port zur Verfügung gestellte Interface. Die Sicht auf den User ist stark abstrahiert.

JDBC: Durch diesen Anschluss ist die Verbindung zu der Datenbank erreichbar. Er erlaubt die Kommunikation zwischen Java und der in der H2 Datenbank laufenden Sprache.

Data: Data ist ein von Persistence zur Verfügung gestelltes Interface zur Kommunikation mit der Businesslogic und den Controllern.

Update: Durch diesen Anschluss wird die Bearbeitung der Dateien an der Persistence ermöglicht. Er wird von Model zur Verfügung gestellt und von Persistence und Businesslogic genutzt.

Control: Durch diesen Anschluss bekommt der Controller seine geforderten Daten von Businesslogic und Model.

HTTPS/Request/Response: Jede Interaktion mit dem System wird per HTTPS gemacht, also sind diese Anschlüsse verantwortlich für die Verschiebung der Dateien von den Webbrowser und den REST-Client zu dem Server und umgekehrt.

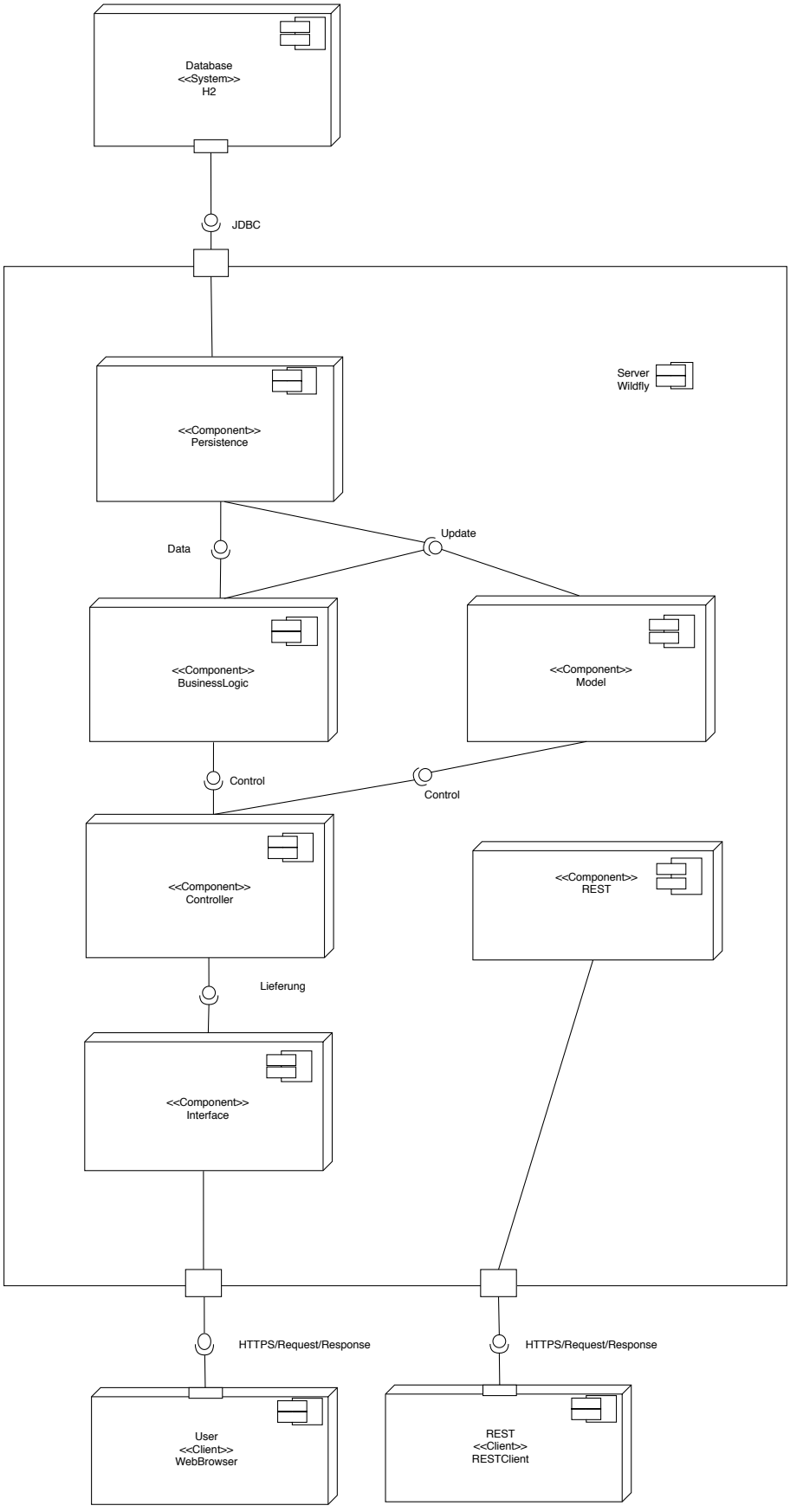

Abbildung 1: Die konzeptionelle Sicht

5 Modulsicht

In diesem Kapitel gehen wir genauer auf die in der konzeptionellen Sicht gezeigten Komponenten ein. Eine detaillierte Beschreibung der Klassen und ihrer Schnittstellen ist im Javadoc der Schnittstellenbeschreibung enthalten. Die Modularisierung ergibt sich aus den Strategien 11.2 und 15.1. Von folgenden Strategien sieht man hier die Umsetzungen, die werden aber nicht näher in den unten folgenden Modulen beschrieben:
In ProzessSchrittParameterController findet sich eine exportJSON Methode, die die Umsetzung von Strategie 4.2 darstellt. Die Strategie 5.1 findet sich in den Methoden in den Klassen AuftragsLogController und ProzessSchrittLogController, durch die die einzelnen Daten der Protokolle einstellbar sind. Die Logs werden durch Methoden in AuftragController und ProzessSchrittController gesetzt. Der Technologe kann durch die Methode prioSort die Aufträge automatisch nach Ihrer Priorität sortiert anzeigen lassen, was der Strategie 7.2 entspricht. Die Strategie 10.2 zeigt sich hier vor allem durch Methoden des ProzessKettenAdministrators wie zum Beispiel setBedingung. Das lazy loading von Strategie 12.1 kann schlecht explizit in der Architektur gesehen werden. Das Laden der Proben wird über die Methoden getProben des Logistikers gemacht. Der Technologe kann durch assignToAuftrag und getAuftrag mit den Transportaufträge umgehen. Mithilfe der Klasse TransportAufträge werden diese Aufträge gemanaged: Hier können unter anderem die Zustände gesetzt werden. Dadurch ist diese Klasse die Umsetzung der Strategie 13.1. Durch die Methode setEs ohne Parameter aus der Klasse ProzessSchrittVorlageController kann einem ProzessSchritt automatisch die beste ExperimentierStation zugeordnet werden. Das passiert unter anderem mithilfe der Bedingungen von Experimentierstationen, und den Eigenschaften von Proben. Dadurch ist die Strategie 14.2 umgesetzt. Der Prozesskettenadmin kann einen Prozessschrittzustandsautomat setzen, was er mithilfe einer seperaten Klasse für diesen Zustandsautomaten(ProzessSchrittZustandsAutomat) macht. Das ist Strategie 19.3.

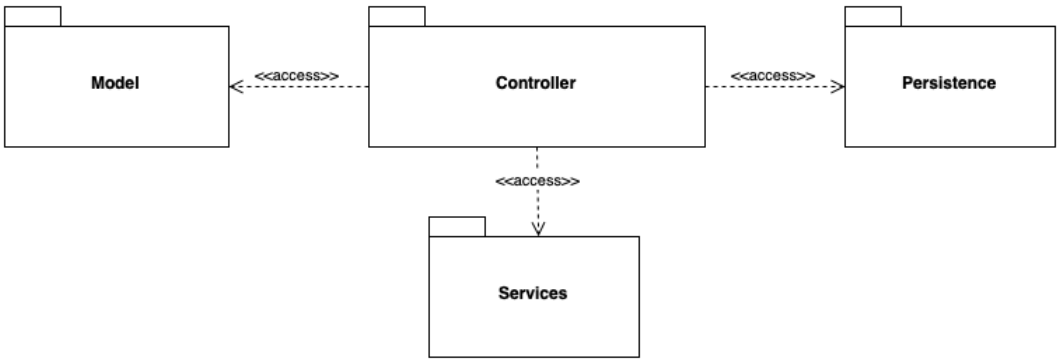

Abbildung 2: Modul Sicht

In den nächsten Abschnitten werden die in der konzeptionellen Sicht gezeigten Module näher beschrieben. Da die Funktion der Module sehr komplex ist, haben wir sie der Übersicht halber in verschiedene Pakete unterteilt, in denen wir näher auf die Funktion eingehen.

5.1 Pakete

In diesem Unterkapitel werden wir jedes Paket der Software detailliert modellieren und erklären. Wir fangen zudem erst einmal mit unserem Modeldiagramm an, welches die Relationen zwischen den Datenklassen zeigt. Dieses Diagramm ist eine verfeinerte Version der Datensicht welche wir in Kapitel 6 modellieren und beschreiben.

5.1.1 Model

Das folgende Diagramm beschreibt eine verfeinerte Version unserer Datensicht (siehe Kapitel 6). Dieses Diagramm beschreibt die Datenklassen unserer Software mit ihren Attributen und Relationen zueinander. Wie man sehen kann, ist es möglich, über den User an die meisten Module des Systems zu gelangen, allerdings kann auch auf die meisten Einzelmodule auch beliebig zugegriffen werden. Der User hat hierbei allerdings eine zentrale Rolle, da er z. B. Prozesskettenvorlagen erstellen kann, und diese nicht von außen erstellt werden können sollen.

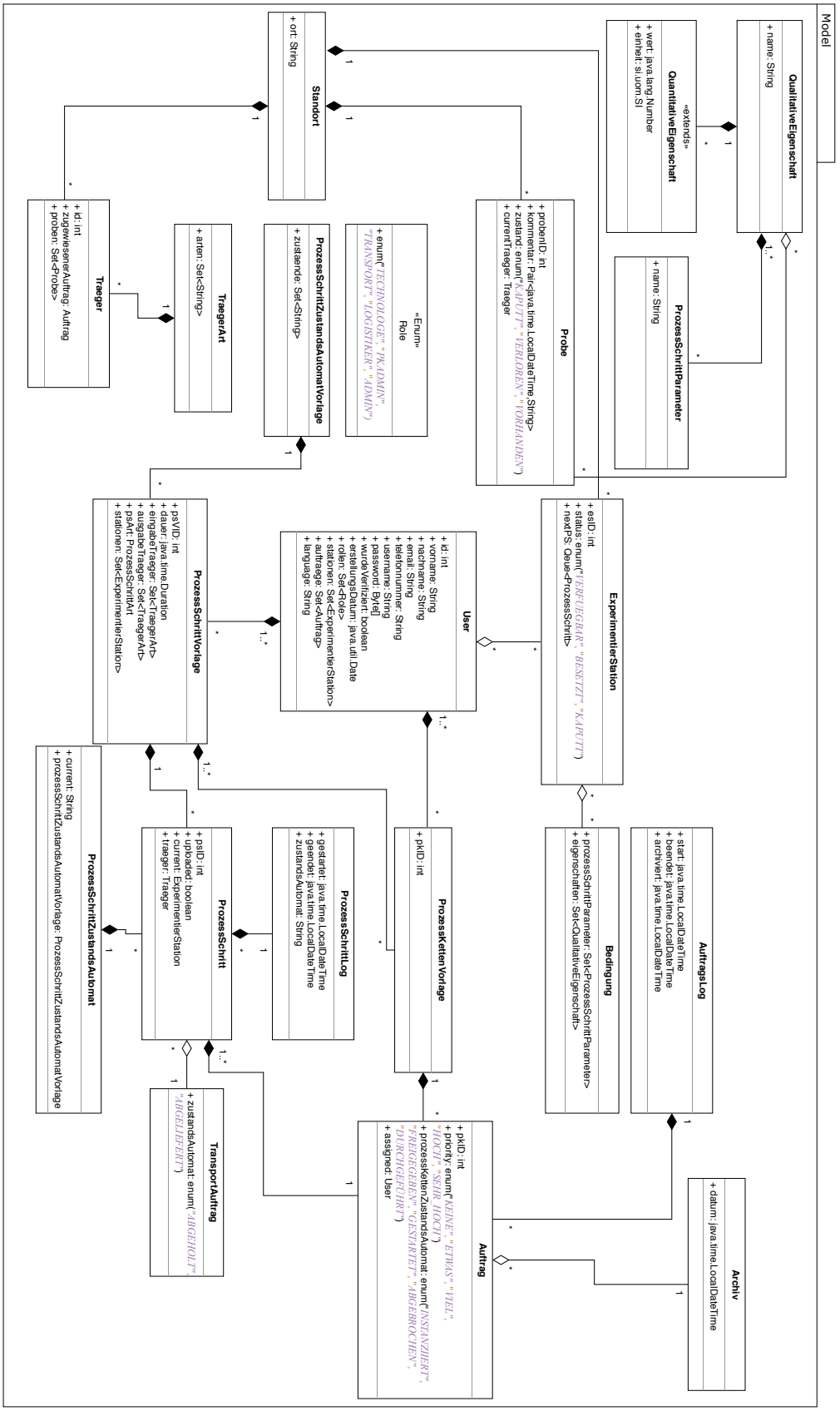

Abbildung 3: Model Sicht

5.1.2 Controller

Nun möchten wir die Controller in unserem System betrachten. Diese Module sind für die Logik zuständig, welche von der WebUI ausgelöst wird. Möchte zudem z.B. ein Prozess Ketten Administrator eine Neue Prozess Ketten Vorlage erstellen, so darf die WebUI nicht direkt auf die Datenklassen verweise machen, sondern muss diese Instruktion erst einmal an einen Controller weitergeben welcher diese anschließend realisiert. Zudem haben diese Controller auch zugriff auf die Persistenz Klassen, welche wir zunächst betrachten werden, um Objekte die von einer Datenklasse durch den Controller erstellt wurden auch in der Datenbank abspeichern zu können.

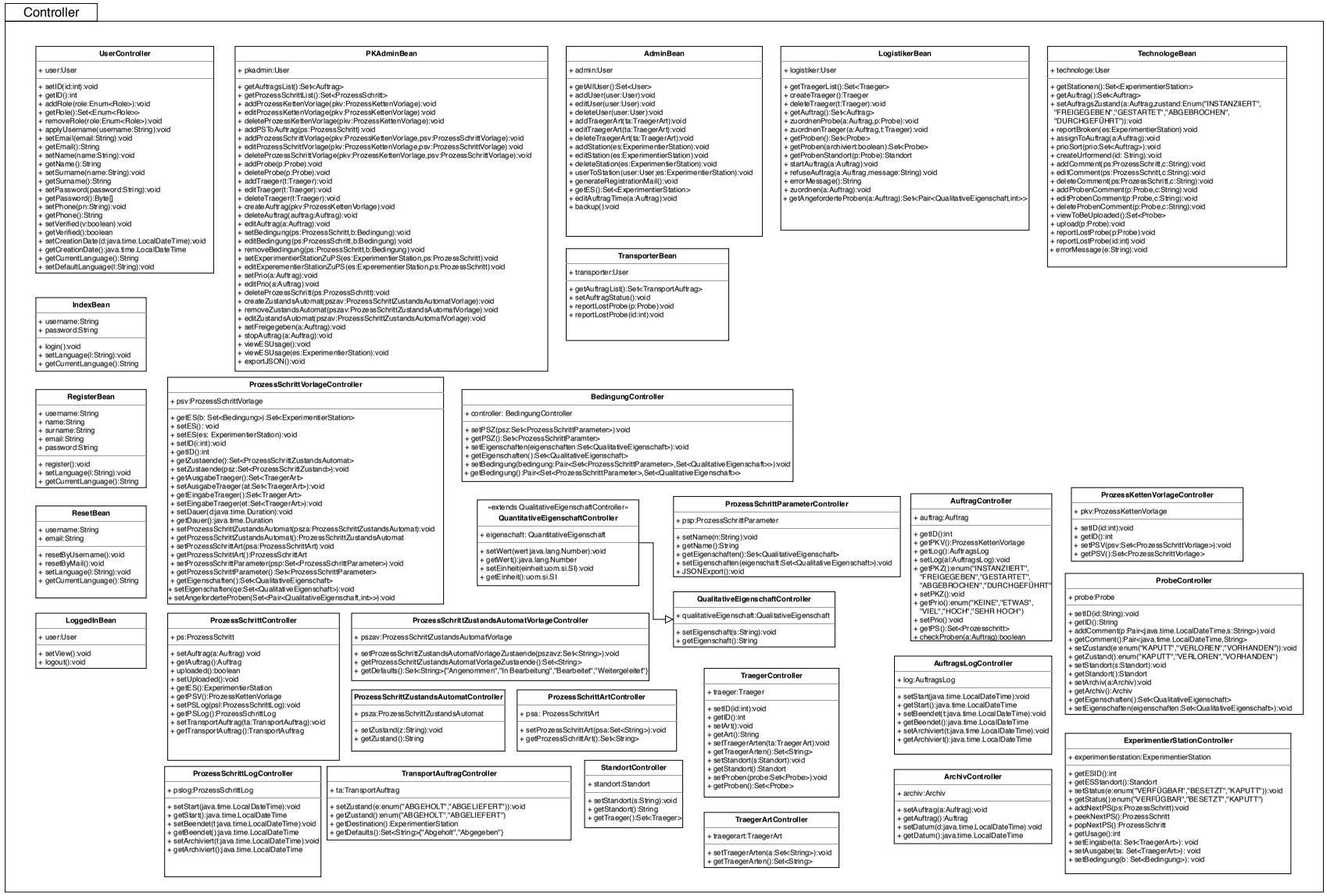

Abbildung 4: Controller Sicht

5.1.3 Persistence

Die Persistenzklassen sind für die Persistierung von Daten zuständig. Sie erben alle von der abstrakten Klasse ObjectDAO. Diese Klasse implementiert den EntityManager, der für das Verwalten der Datenbankinformationen zuständig ist. Die verschiedenen Funktionen sollen zum Beispiel das Speichern und das Laden nach einem Neustart von Daten ermöglichen. Jedes Objekt, was für die Speicherung in der Datenbank relevant ist, hat eine Klasse, die von ObjectDAO erbt. ObjectDAO hat drei abstrakte Funktionen, die jeweils in den Klassen für die einzelnen Objekte implementiert werden. Zusätzlich gibt es in zu jeder Klasse die relevanten Get Funktionen, um Informationen aus diesen Klassen zu bekommen. Die Funktion persist(T) speichert Objekte in die Datenbank. Update() überschreibt eine vorherige Version dieses Objektes mit der neuen und remove() entfernt das Objekt.

Dieses Paket wird gebraucht, da es vorgegeben ist.

Abbildung 5: Persistenzmodelll für FarbigesSFB

5.1.4 Services

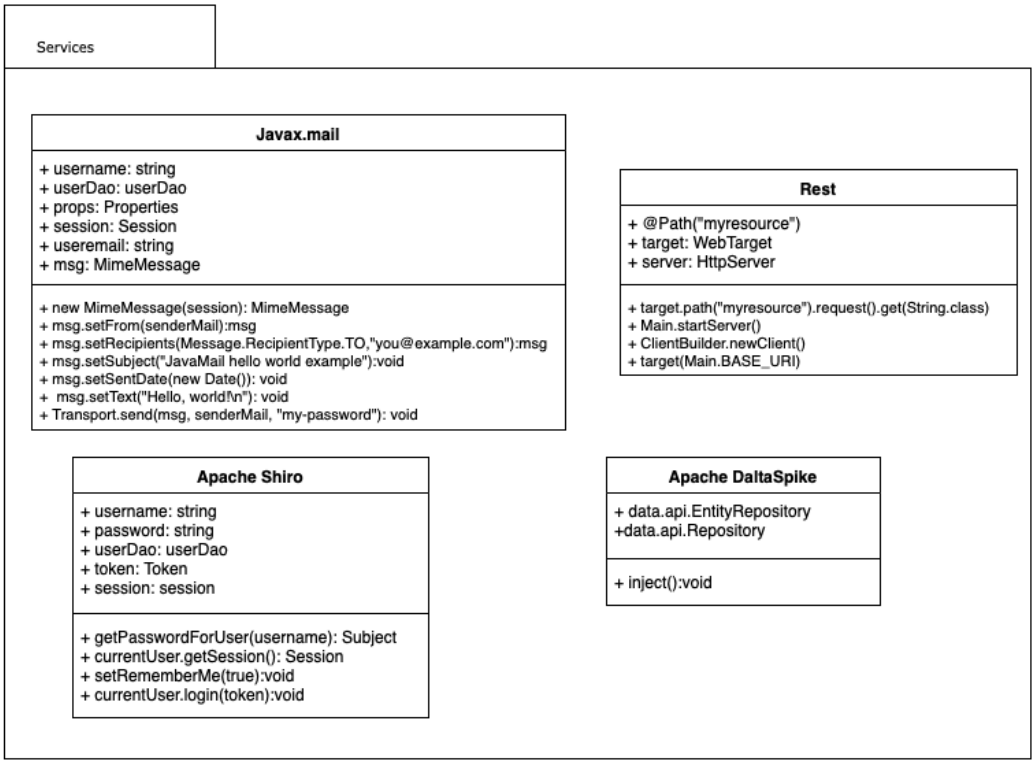

Abbildung 6: Services

Dieses Diagramm beschreibt die in unserem System verwendeten Services. Unser System soll z. B. bei der Erstellung von Benutzern E-Mails versenden können, wozu wir das E-Mail Modul haben. Zugleich soll unser System durch Apache Shiro gesichert sein, wodurch wir das Modul Apache Shiro haben. Wir möchten zur Datenverwaltung außerdem noch Deltaspike Data benutzen, wodurch wir auch dieses Modul haben.
Hier werden unter anderem die Strategien 2.1 und 6.1 umgesetzt.

5.2 Module

5.2.1 User Modul

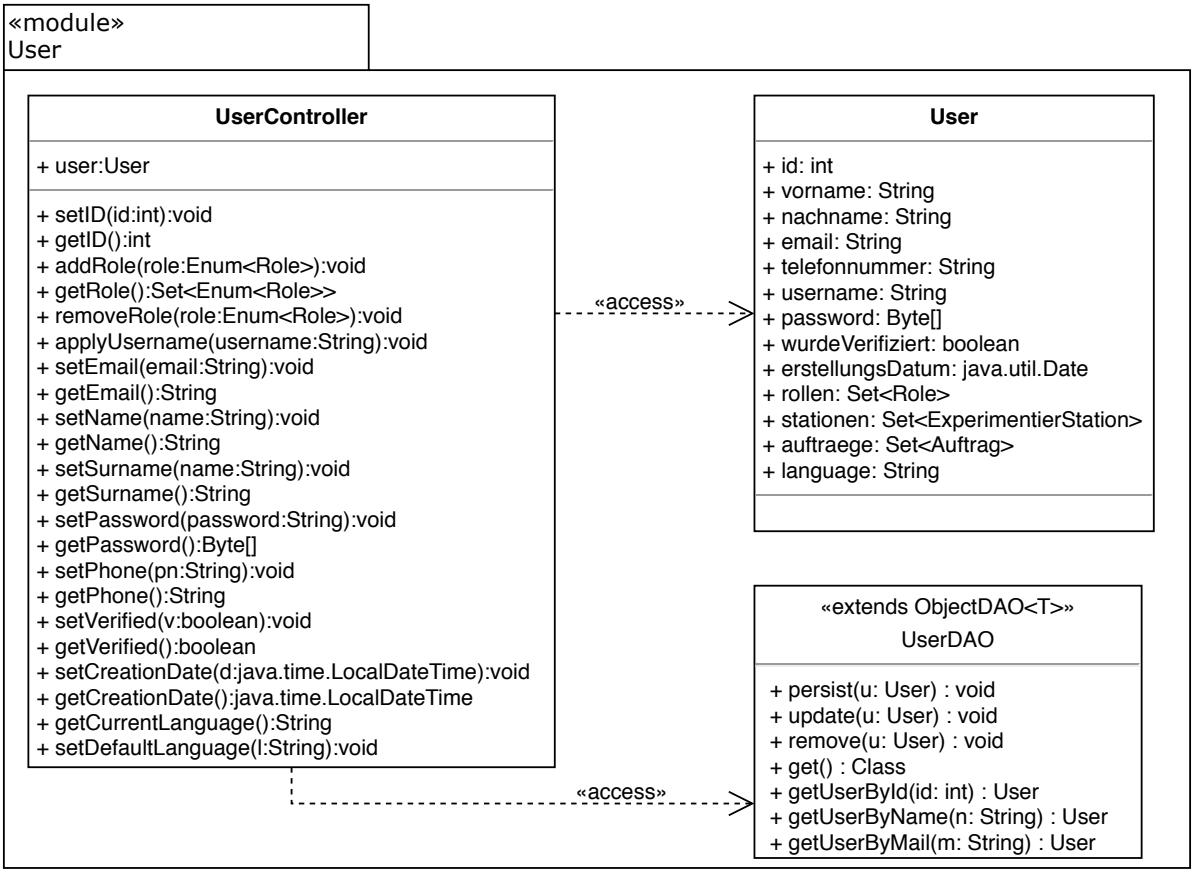

Abbildung 7: User Module

Um User erstellen oder verwalten zu können, muss unser UserController Zugriff auf die Userdatenklasse haben. Zusätzlich möchte man die User in der Datenbank einspeichern und verwalten können, wozu der Controller auch Zugriff auf die DAO Klasse von User hat. Jeder User hat eine eindeutige ID, und muss mindestens eine Rolle zugewiesen haben.
Hier sieht man die Umsetzung der Strategie 1.2 anhand der Methoden setRole, und getRole, wobei letztere ein Set aus mehreren Rollen zurückgeben kann.

5.2.2 Prozesskettenvorlage Modul

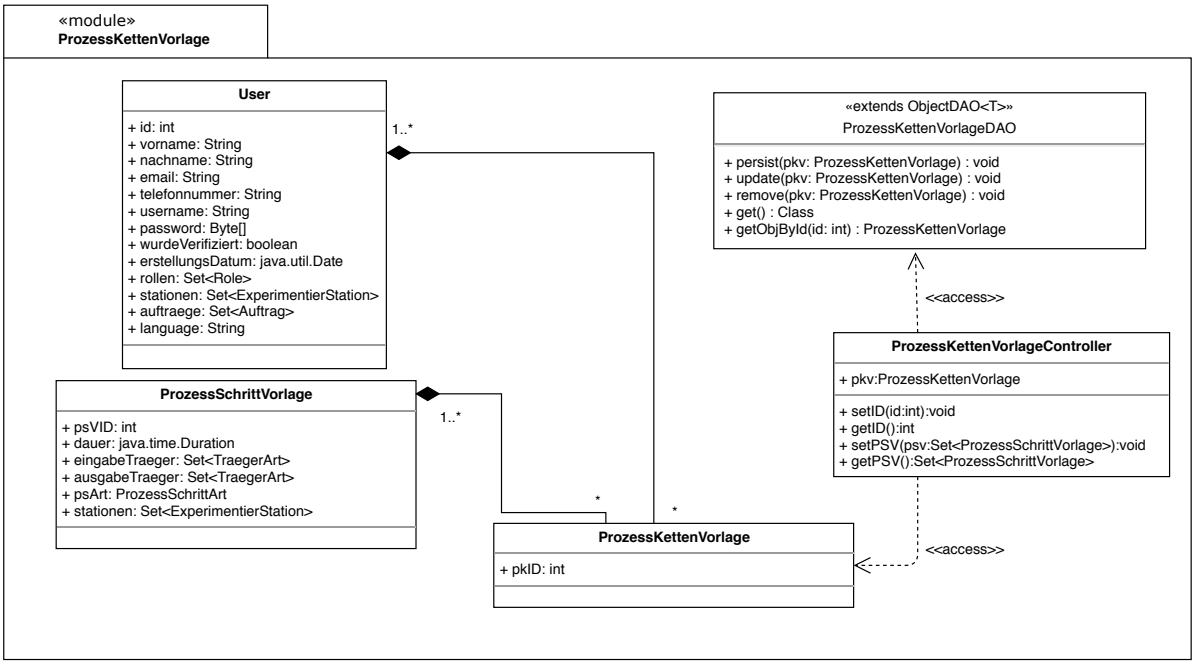

Abbildung 8: Prozess Ketten Vorlage Module

Um Prozesskettenvorlagen erstellen und verwalten zu können, muss der ProzessKettenController Zugriff auf die zugehörige Datenklasse haben. Zusätzlich sollen die Prozesskettenvorlagen in der Datenbank gespeichert und verwaltet werden, wozu der Controller auch Zugriff auf die DAO Klasse haben muss. Eine Prozesskettenvorlage braucht mindestens einen User, der der Prozesskettenadministrator ist, der diese Vorlage erstellt hat, und besteht aus mindestens einer Prozessschrittvorlage.
Hier wird Strategie 3.1 für die Prozessketten umgesetzt.

5.2.3 Auftrags Modul

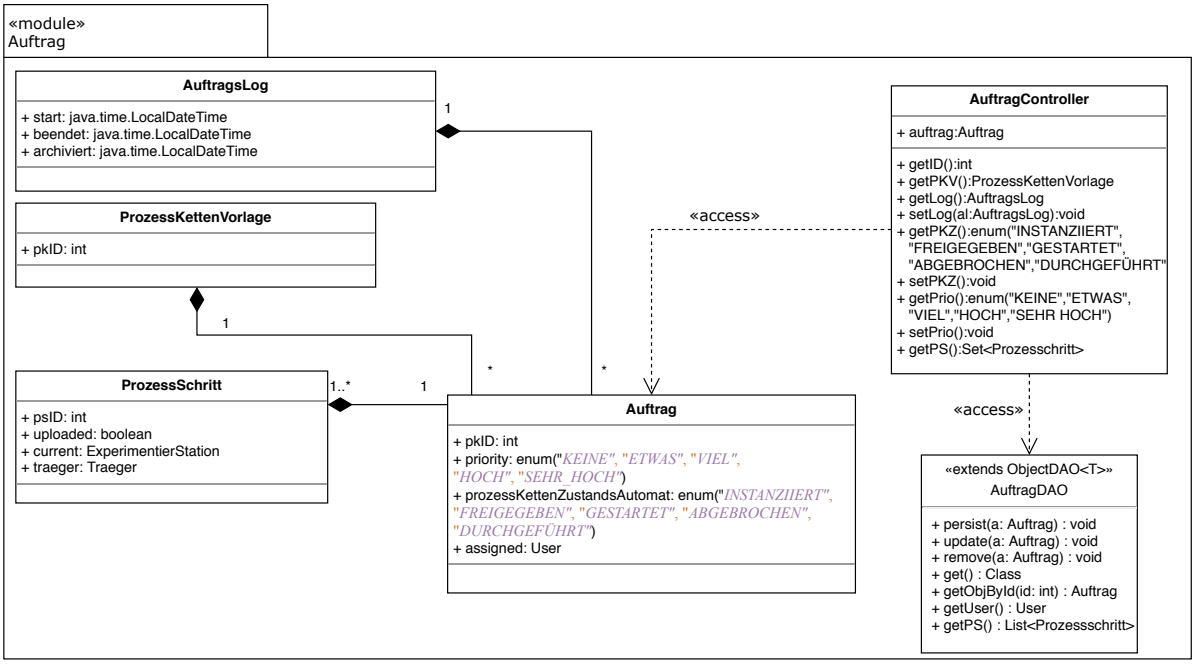

Abbildung 9: Auftrag Module

Um nun Aufträge erstellen oder verwalten zu können, muss unser Auftragscontroller Zugriff auf die Auftragsdatenklasse haben. Zusätzlich möchte man diese in der Datenbank einspeichern und verwalten können, wozu der Controller auch Zugriff auf die DAO Klasse von Auftrag hat.

5.2.4 Prozessschrittvorlage Modul

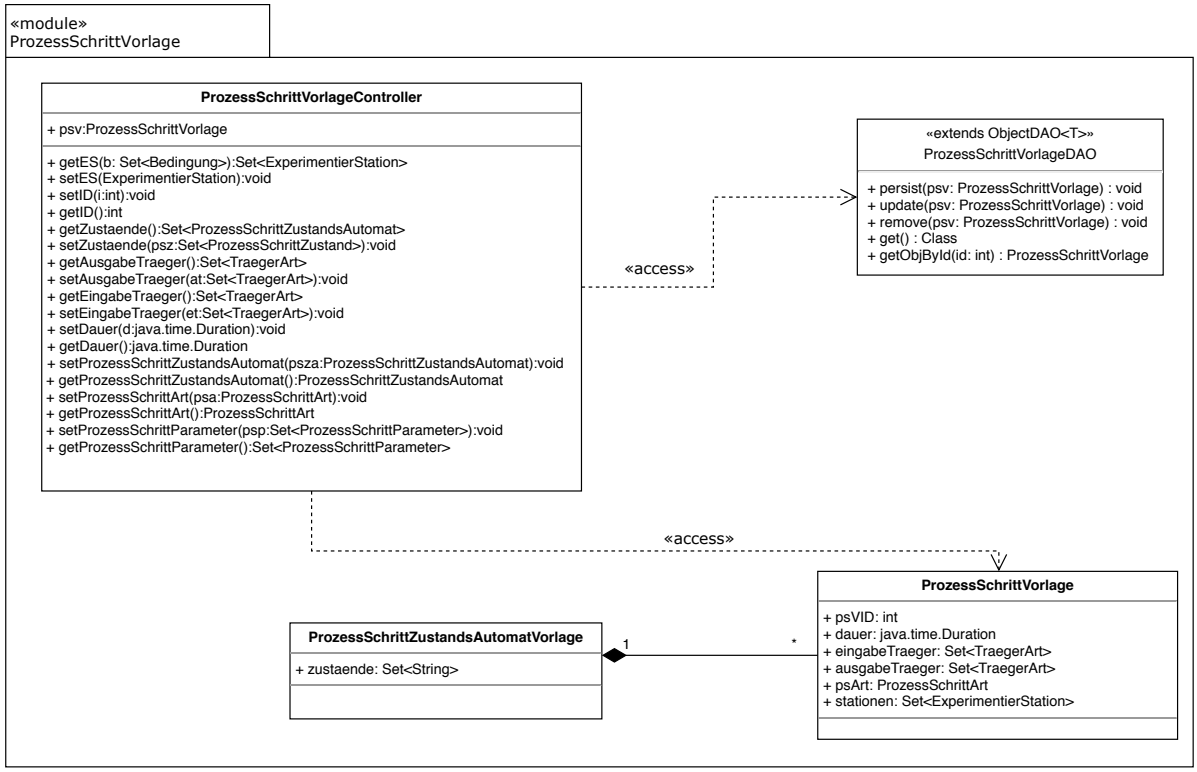

Abbildung 10: Prozessschrittvorlage Module

Um Prozessschrittvorlagen erstellen zu können, benötigen wir allerdings Prozessschrittvorlagen, aus welchen letztendlich Prozessschritte eines Auftrags erstellt werden. Hierfür dient das Modul Prozessschrittvorlage. Dieses Modul ist für die Erstellung und Bearbeitung von Prozessschrittvorlagen zuständig. Um nun Prozessschrittvorlagen erstellen zu können, benötigen wir erst einmal einen Prozessschrittzustandsautomat welcher den Zustand eines Prozessschritts darstellt. Dieser wird allerdings von einer Vorlage erstellt, welche in der Prozessschrittvorlage abgespeichert wird, wodurch der Prozessschritt, der aus dieser Vorlage erstellt wird, anschließend weiß welche zustände er haben kann. Um nun Prozessschrittvorlagen erstellen zu können, muss unser Controller auf die Datenklasse Prozessschrittvorlage Zugriff haben. Um diese Vorlagen anschließend in der Datenbank verwalten zu können, muss er auch Zugriff auf die DAO Klasse von Prozessschrittvorlage haben.
Hier wird Strategie 3.1 für die Prozessschritte durchgesetzt.

5.2.5 Prozessschritt Modul

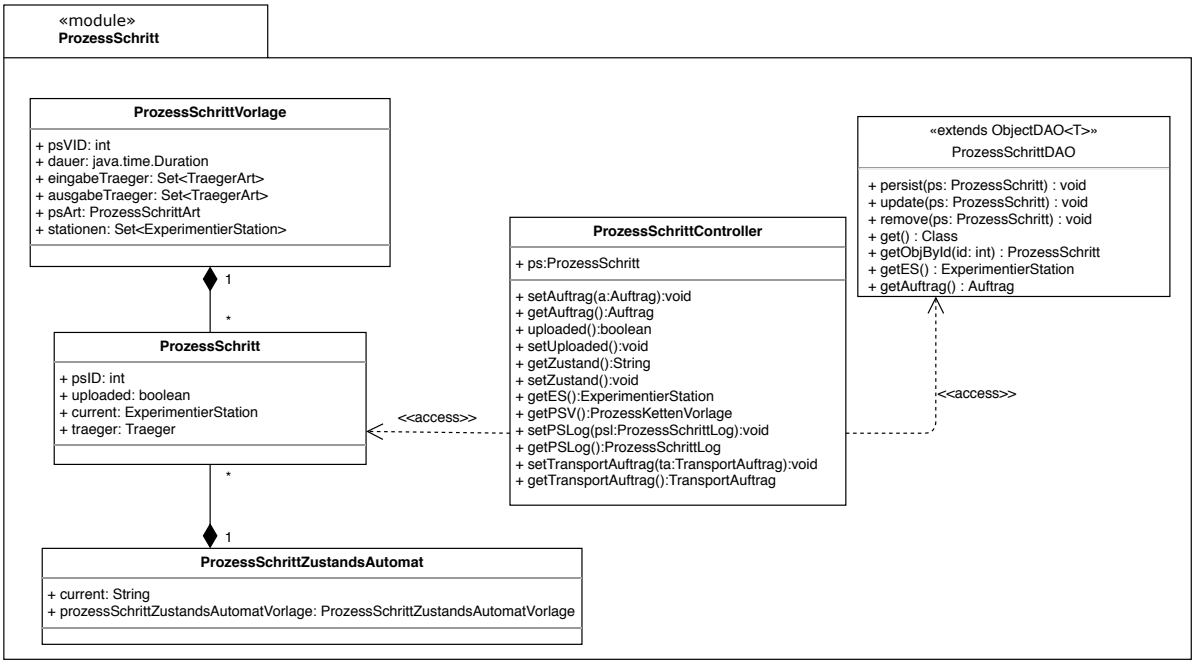

Abbildung 11: Prozessschritt Module

Um Prozessschritte erstellen und verwalten zu können, braucht der ProzessSchrittController Zugriff auf die ProzessSchritt Datenklasse. Da die Prozessschritte auch in der Datenbank gespeichert und verändert werden sollen, muss der Controller auch Zugriff auf die DAO haben. Ein Prozessschritt besteht aus einer ProzessSchrittVorlage, und hat einen ProzessSchrittZustandsAutomat.

5.2.6 Probe Modul

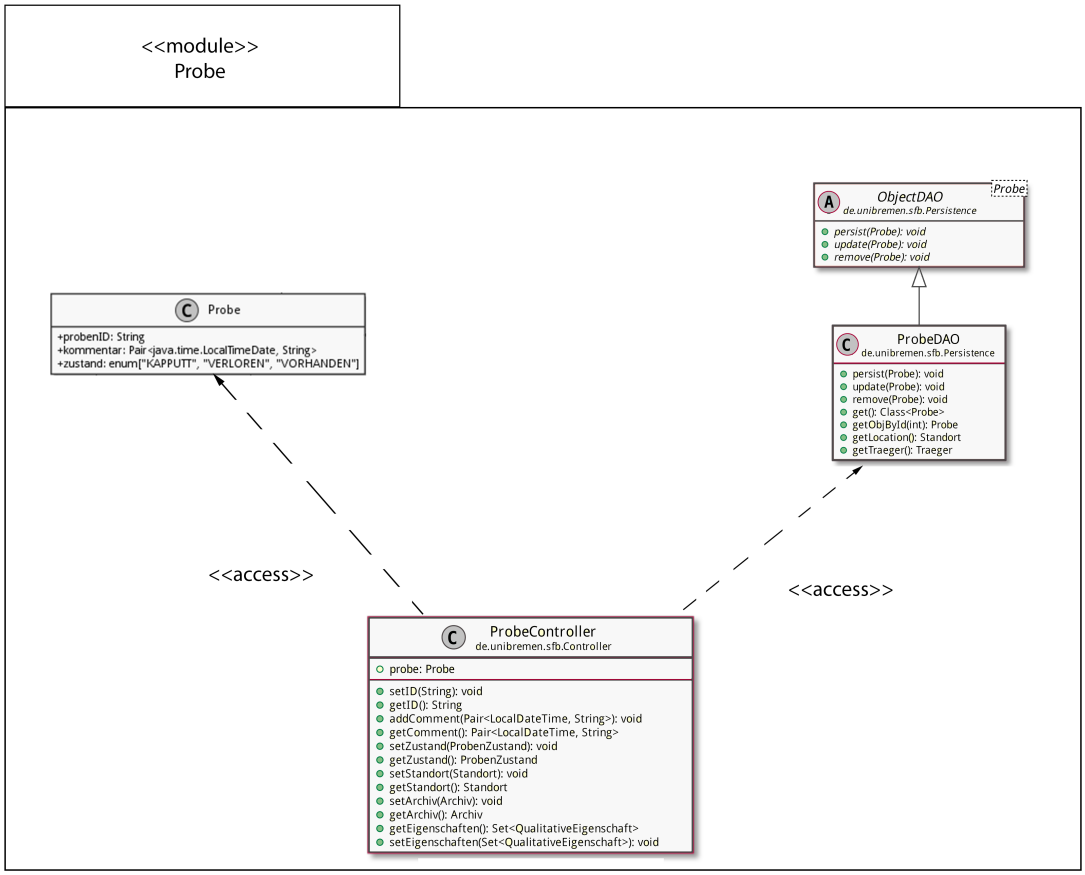

Abbildung 12: Probe Module

Um Proben verwalten und erstellen zu können, braucht der ProbeController Zugriff auf die Probendatenklasse. Da Proben auch in der Datenbank gespeichert und verändert werden sollen, muss der Controller auch Zugriff auf die DAO Klasse haben. Jede Probe hat eine eindeutige ID, und befindet sich in einem Zustand.
Die Umsetzung der Strategie 8.1 zeigt sich vor allem dadurch, dass es keine Methoden oder andere Aspekte der Architektur gibt, die irgendwie Proben in Gruppen sortieren. Jede Probe hat eine ID. Des weiteren kann jede Probe durch Ihre ID identifiziert werden (so zum Beispiel in der Methode reportLostMethode des Technologen).

5.2.7 Träger Modul

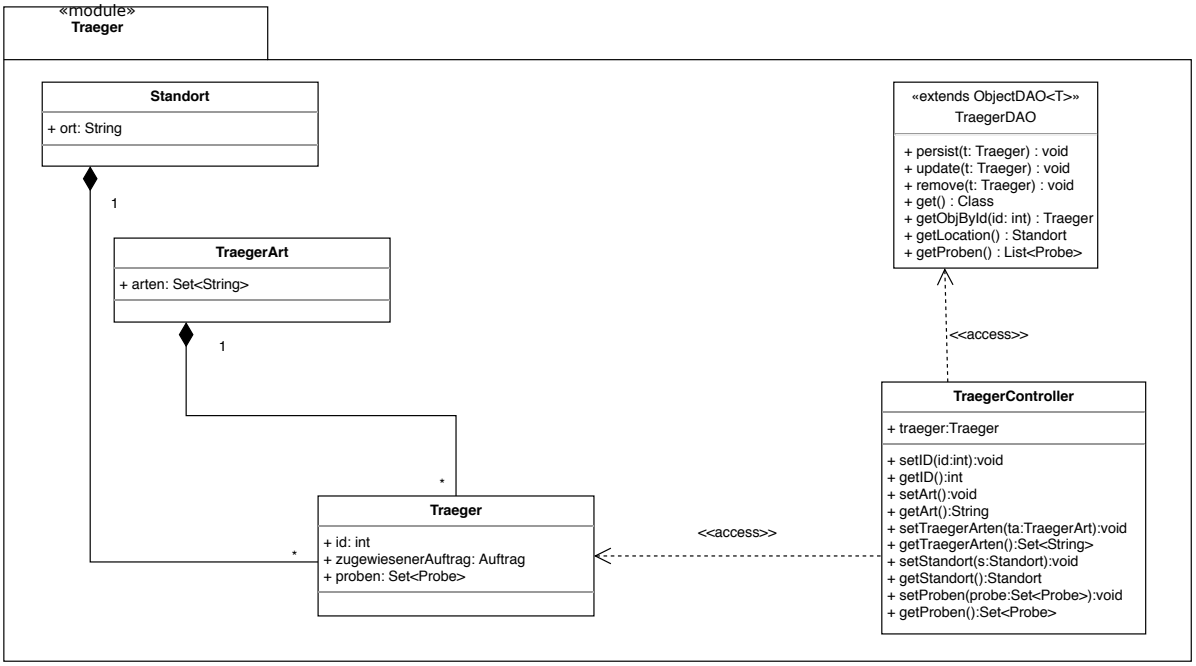

Abbildung 13: Träger Module

Die Proben in unserem System werden in Trägern transportiert. Diese Träger benötigen einen Standort zu jeder Zeit und haben immer eine Art. Um nun diese Träger verwalten zu können braucht unser Controller Zugriff auf die Träger Datenklasse. Um diese dann in der Datenbank verwalten zu können, braucht er ebenfalls Zugriff auf die TraegerDAO klasse.
Strategie 17.3 wird hier umgesetzt.

5.2.8 Experimentier Station Modul

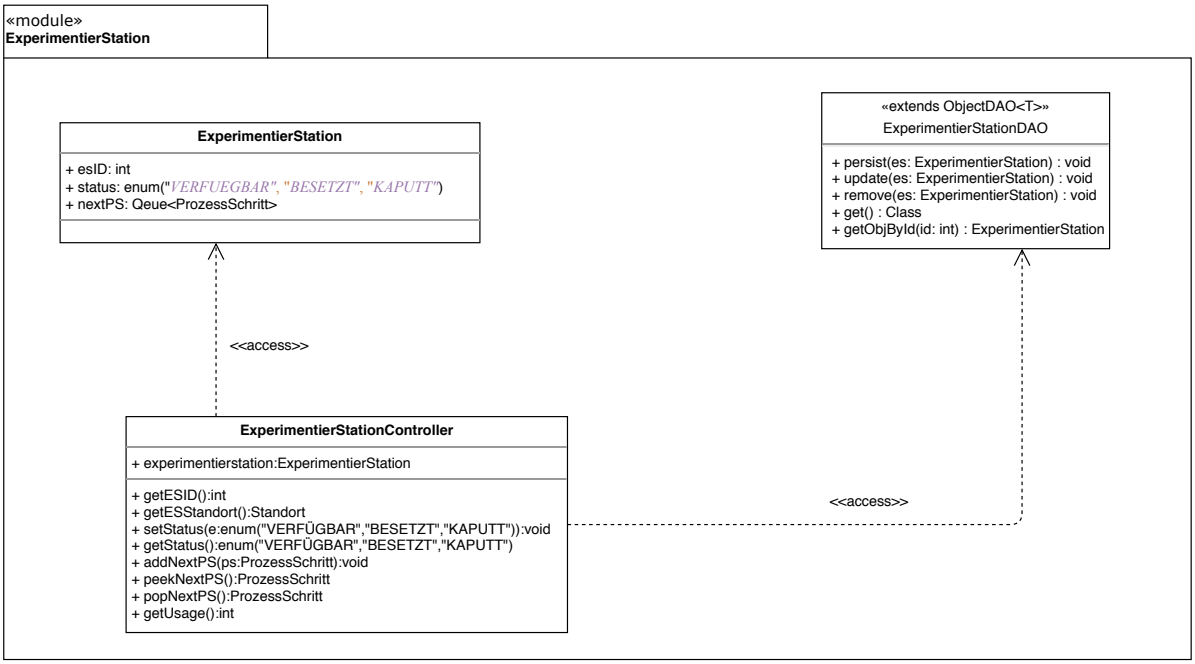

Abbildung 14: Experimentierstation Module

Um die Experimentierstationen in unserem System erstellen und verwalten zu können, braucht der ExperimentierStationController Zugriff auf die Datenklasse Experimentier-Station. Da Experimentierstationen auch in der Datenbank gespeichert und verändert werden sollen, braucht der Controller auch Zugriff auf die DAO Klasse. Eine Experimentierstation hat eine eindeutige ID, befindet sich immer in einem Zustand, befindet sich an einem Standort, und hat eine Warteschlange mit Prozessschritten, die abgearbeitet werden sollen.

5.2.9 Prozessschritt Zustandsautomat Vorlage Modul

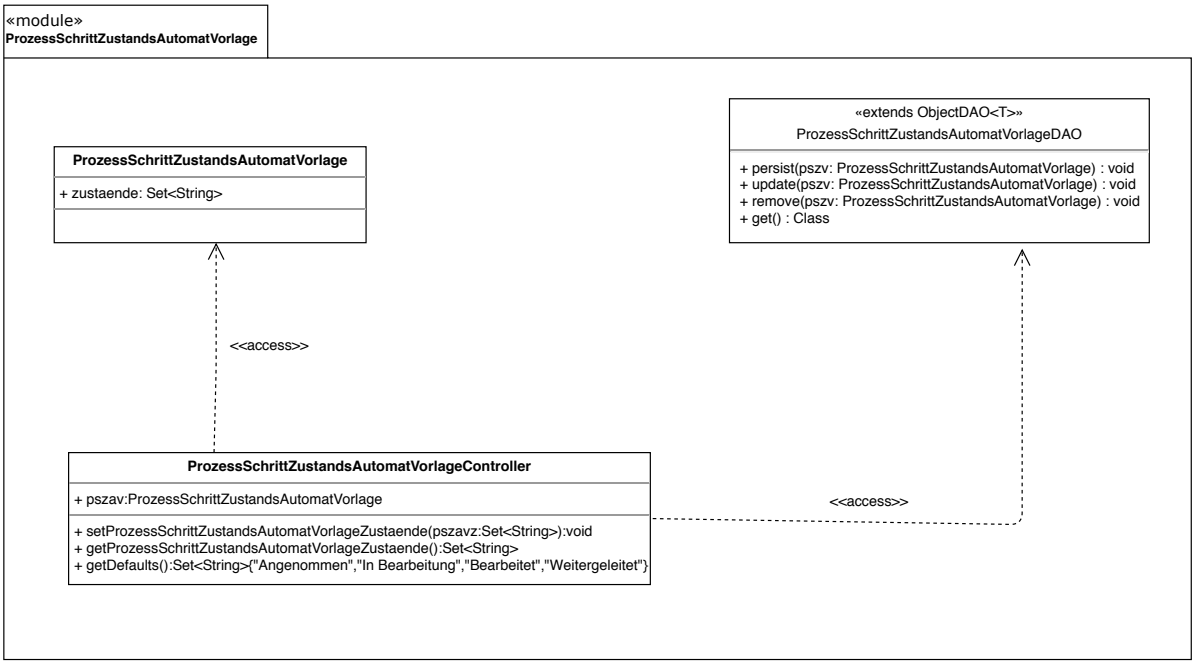

Abbildung 15: Prozess Schritt Zustands Automat Vorlage Module

Um Prozessschrittzustandsautomatvorlagen in unserem System erstellen und verwalten zu können, braucht der ProzessSchrittZustandsAutomatVorlageController Zugriff auf die Datenklasse ProzessSchrittZustandsAutomatVorlage. Da die Vorlagen auch in der Datenbank gespeichert und verändert werden sollen, braucht der Controller auch Zugriff auf die zugehörige DAO Klasse.
Hier wird die Strategie 19.3 umgesetzt.

5.2.10 Prozessschritt Zustandsautomat Modul

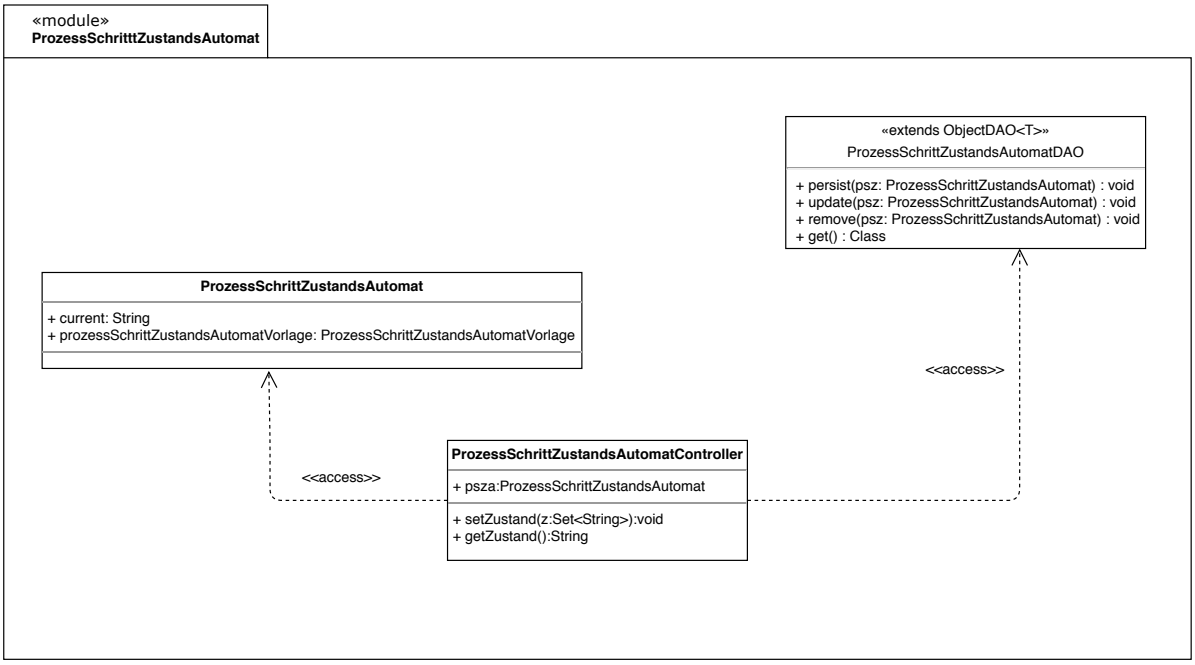

Abbildung 16: Prozess Schritt Zustands Automat Module

Um Prozessschrittzustandsautomaten in unserem System erstellen und verwalten zu können, braucht der ProzessSchrittZustandsAutomatController Zugriff auf die Datenklasse ProzessSchrittZustandsAutomat. Da die Zustandsautomaten auch in der Datenbank gespeichert und verändert werden können, braucht der Controller auch Zugriff auf die zugehörige DAO Klasse. Ein Prozesskettenzustandsautomat befindet sich immer in einem festen Zustand.
Hier wird Strategie 19.3 umgesetzt.

5.2.11 Bedingung Modul

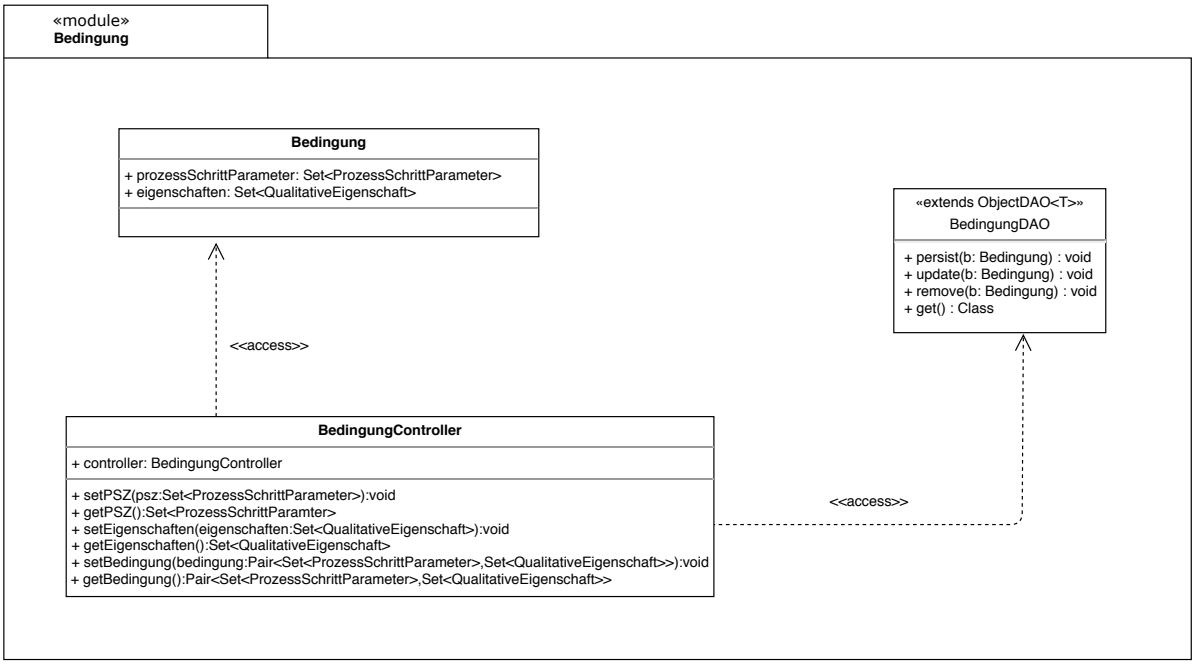

Abbildung 17: Bedingung Module

Um Bedingungen in unserem System erstellen und bearbeiten zu können, braucht der BedingungController Zugriff auf die Datenklasse Bedingung. Da Bedingungen auch in der Datenbank gespeichert und verändert werden müssen, soll der Controller auch Zugriff auf die zugehörige DAO Klasse haben.
Dies wird für die Umsetzung von Strategie 14.2 benutzt.

5.2.12 Prozessschritt Log Modul

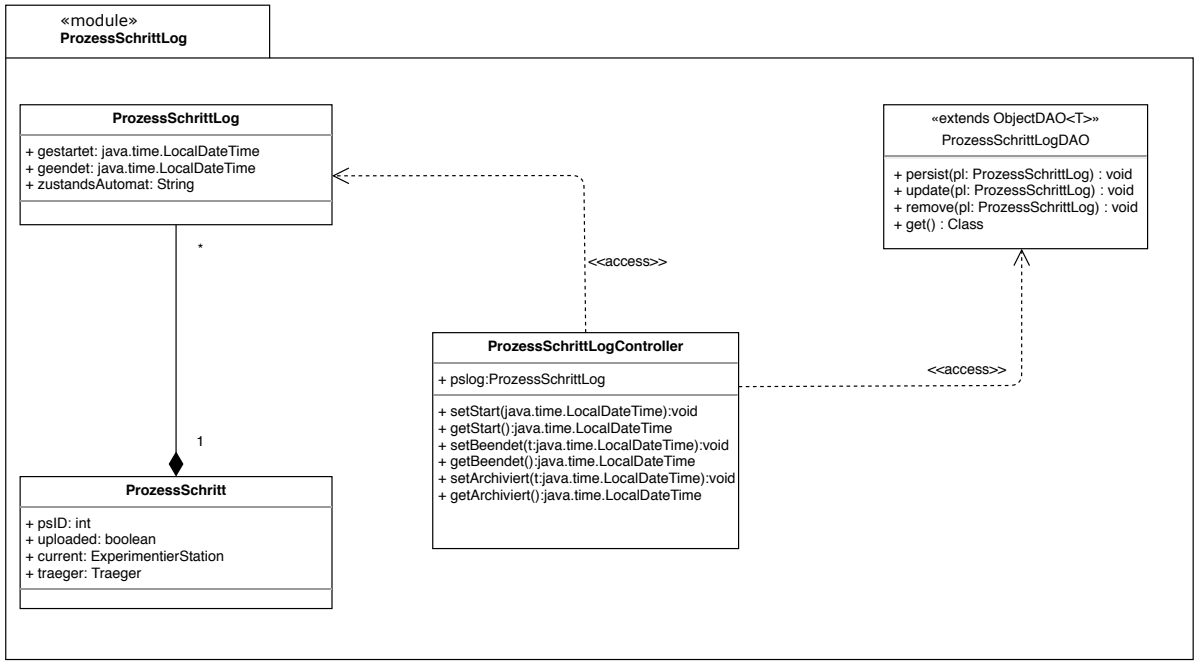

Abbildung 18: Prozess Schritt Log Module

Um Prozessschrittprotokolle in unserem System erstellen und verändern zu können, braucht der ProzessSchrittLogController Zugriff auf die Datenklasse ProzessSchrittLog. Da die Protkolle auch in der Datenbank gespeichert und verändert werden sollen, braucht der Controller auch Zugriff auf die DAO Klasse. Jedes Protokoll gehört zu genau einem Prozessschritt.
Hier wird Strategie 5.1 für die Prozessschritte umgesetzt. Die komplette Umsetzung besteht daraus, dass für jeden Prozessschritt ein Log erstellt wird, und für jeden Auftrag die Prozessschrittlogs und die Auftragslogs zusammen das Protokoll ergeben.

5.2.13 Transport Auftrag Modul

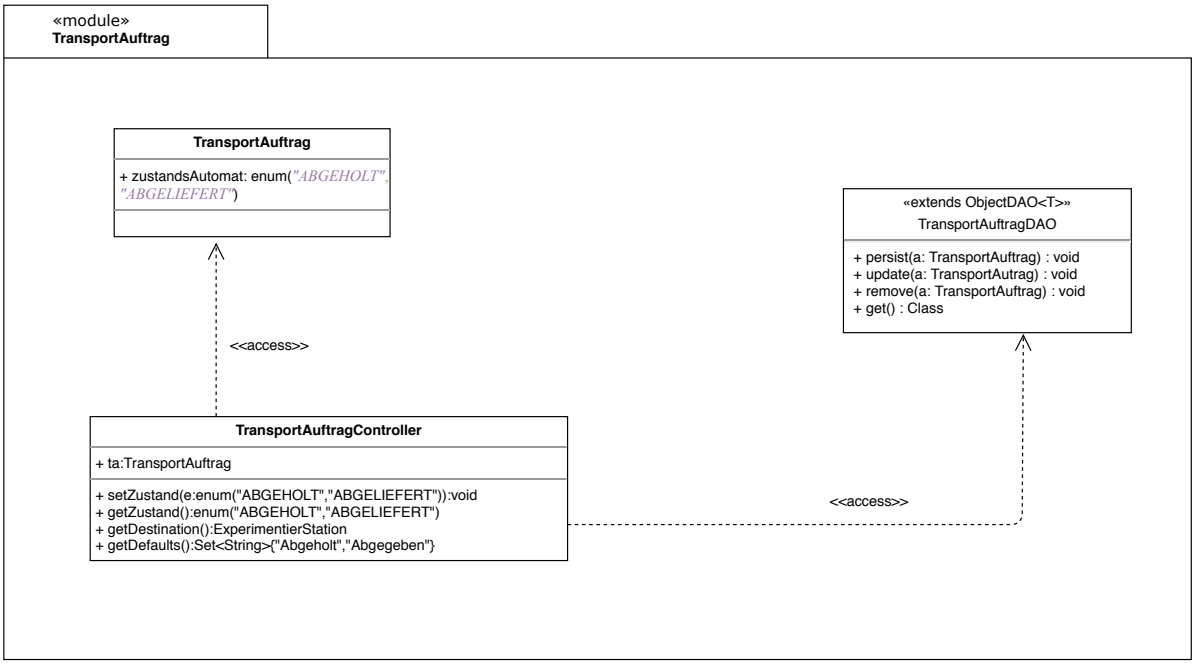

Abbildung 19: Transport Auftrag Module

Um Transportaufträge in unserem System erstellen und bearbeiten zu können, braucht der TransportAuftragController Zugriff auf die Datenklasse TransportAuftrag. Da die Transportaufträge auch in der Datenbank gespeichert und verändert werden sollen, braucht der Controller auch Zugriff auf die DAO Klasse. Jeder Transportauftrag hat einen Zustandsautomat.
Hier setzen wir Strategie 13.1 um.

6 Datensicht

Alle Rollen, die ein Nutzer annehmen kann, sind in einer Enum gespeichert. Diese sind TECHNOLOGE, Prozesskettenadministrator, TRANSPORT, LOGISTIKER und der ADMIN. Hiermit realisieren wir die Strategie 1.2 3.2. Wir können somit mehrere Rollen für einen User speichern. Um die Datensicherheit zu gewährleisten, haben wir uns für Apache Shiro entschieden. Hiermit können wir einzelne Programmfunktionen nur für bestimmte Rollen zulassen. Damit setzen wir die Strategie 2.1 3.2 um. Apache Shiro muss wissen, welche Rollen zu welchen Usern gehören. Daher referenziert User eine oder mehrere Rollen.

Die User bestehen aus einer ID, einem Vornamen, einem Nachnamen, einer E-Mail-Adresse, einer Telefonnummer und einer Kombination von seinem Passwort und seinem Usernamen. Der User erhält, nach dem sein Konto erstellt wurde, eine E-Mail, mit der er sich verifizieren muss. Nachdem er seinen Account verifiziert hat, wird wurdeVerifiziert auf True gesetzt. Das Attribut Erstellungsdatum speichert, wann der Nutzer erstellt worden ist.

Ein Standort enthält ein String, in dem der Ort des Standorts steht.

Ein Träger hat eine ID und einen Standort. Falls er einem Auftrag zugewiesen ist, referenziert er diesen auch.

Es gibt standardmäßig die Trägerarten eingebettet, einzelne und Glas und diese können später nach Belieben des Kunden durch neue Trägerarten erweitert werden. Die Trägerarten sind in der Klasse Traegerart gespeichert. Die Information über die Trägerarten wird benötigt, um in Erfahrung zu bringen, ob eine Experimentierstation einen Prozessschritt ausführen kann. Die ProzessschrittVorlage hat als Attribut welche Träger sie annimmt und welche Träger sie ausgibt (eingabeTräger, ausgabeTräger). Siehe Strategie 17.3 3.2.

Wir haben das Archiv als eigenständig Klasse modelliert. So können wir im Nachhinein Proben, welche archiviert werden, eine Referenz zu einer Instanz von Archiv geben. Falls es diese Referenz gibt, so können wir wissen, dass diese Probe archiviert wurde. Dies wird in der Strategie 16.1 3.2 umgesetzt.

Es gibt zwei Arten von Eigenschaften, die wir modellieren. Die Qualitative Eigenschaft beschreibt, ob das Objekt diese Eigenschaft hat. Sie hat also ein Namen. Falls es eine Referenz dahin gibt, wissen wir, dass das Objekt auch diese Eigenschaft hat. Die Quantitative Eigenschaft ist eine Erweiterung der Qualitativen Eigenschaft. Sie enthält einen Wert und eine Einheit. Der Wert ist eine Zahl, welche von Java unterstützt wird. Für die Einheiten nutzen wir die SI Implementierung, welche die Java Laufzeit Bibliothek bereitstellt. Dies ermöglicht es unserem System, mit Eigenschaften und Anforderungen umgehen zu können. Es ist uns hiermit möglich, Proben mit spezifizierten Eigenschaften zu finden und diese dann bestimmten Klassen zuzuweisen. Für Experimentierstationen gelten Bedingungen. Diese werden durch Eigenschaften oder Prozess-Parameter realisiert. Wir können festlegen, dass ein Prozessschritt nur ausgeführt werden kann, wenn

ein vorheriger Prozessschritt eine Eigenschaft besitzt oder einen Parameter hat. Ein Logistiker kann in der Logistik-Ansicht Proben auflisten, welche einer Bedingung entsprechen. Hier haben wir die Strategie 10.2 3.2 umgesetzt.

Eine ProzessschrittVorlage enthält eine eindeutige ID, einen Zustandsautomaten (ProzessschrittZustandsAutomat) und die geschätzte Dauer für den Prozessschritt. Strategie 19.2 3.2 setzen wir um, indem wir die Zustände einer ProzessschrittVorlage nicht als Enum speichern, sondern sie als Klasse modellieren. Das Attribut eingabeTraeger ist Liste von Trägerarten, die als Eingabe akzeptiert werden. Die Liste der Trägerarten ausgabeTraeger beschreibt, welche Träger von diesem Prozessschritt ausgegeben werden.

Die Enum AuftragsPriorität enthält die Prioritäten, welche eine Prozesskette haben kann. Falls ein Prozessschritt eine höhere Priorität hat, wird dieser vor dem Prozessschritt, der eine geringere Priorität hat, ausgeführt. Somit kann dann automatisch, unter Berücksichtigung der Priorität, bestimmt werden, welcher Prozessschritt als Nächstes an einer Experimentierstation ausgeführt wird. Hiermit setzen wir die Strategie 7.2 3.2 um.

Es gibt zu Beginn drei Arten von ProzessschrittArten: umformend, färbend und ermittelend. Diese sind vom Kunden vorgegeben und können im Nachhinein nach Belieben des Kunden erweitert werden. Diese wurden als Klasse mit einem Set von Strings modelliert, dies ist unsere Umsetzung der Strategie 20.2 3.2.

Ein Prozessschritt ist eine instanziierte ProzessschrittVorlage. Ihre Attribute sind eine Prozessschritt-ID, ein Boolean uploaded, der auf True gesetzt wird, nachdem der Technologe die ProzessParameter in die externe Datenbank hochgeladen hat, current, welches eine Experimentierstation referenziert, falls es eine Experimentierstation gibt, wo aktuell dieser Prozessschritt durchläuft, und ein String zustandsAutomat, der den aktuellen Zustand aus dem ProzessschrittZustandsAutomaten speichert. Ein Prozessschritt gehört zu einem Auftrag, ein Auftrag kann aber viele Prozessschritte enthalten.

Ein ProzessschrittZustandsAutomat ist eine Liste von Zuständen. Der ProzessschrittZustandsAutomat kann vom Prozesskettenadministrator um weitere Zustände erweitert werden, wenn dieser nicht in Benutzung ist.

Ein ProzessschrittParameter hat einen Namen. Einer ProzessschrittVorlage können Parameter zugewiesen werden. Falls der Prozessschritt ermittelnd ist, wird nachher die Eigenschaft, welche ermittelt wurde, der Probe zugewiesen.

JSON ist der aktuelle De-facto-Datenformat-Standard um Daten über APIs bereitzustellen. Das Java-Ökosystem bietet eine Reihe von Bibliotheken zur Erstellung von JSON aus Java-Objekten und umgekehrt (GSON, Jackson, etc.). Mit dem Release von Java EE 8 und der JSR-367 haben wir nun einen standardisierten Ansatz dafür: JSON-B.

Um den Up/Download von den Parametern der Prozessschritte als JSON zu ermöglichen verwenden wir JSON-B. Hiermit setzen wir die Strategie 4.1 3.2 um. Dafür stellen wir ein Interface bereit, dass es uns ermöglicht die Parameter der Prozessschritte zu serialisieren. Um mit externen Diensten zu kommunizieren, werden für das serialisierte JSON Rest-Schnittstellen angeboten.

Eine ProzesskettenVorlage, hat eine eindeutige ID.

Die Enum ProzesskettenZustandsAutomat enthält alle Zustände, die ein Auftrag haben kann. Diese sind fest und können im Nachhinein nicht mehr geändert werden. Deshalb werden sie als Enum gespeichert. Hiermit wird die Strategie 18.1 umgesetzt.

Ein Auftrag ist eine Instanz einer ProzesskettenVorlage.

Wie oben erwähnt, ist Auftrag eine Instanz von der ProzesskettenVorlage, die ProzessschrittVorlage erstellt ein Prozessschritt Objekt. Wir haben getrennte Klassen für die Vorlage und die Instanzen von den Prozessschritten und Prozessketten. Hiermit setzen wir die Strategie 3.1 3.2 um. Die Prozessschritte sind somit abstrakt und können später auch in anderen Prozessketten wiederverwendet werden.

Ein AuftragsLog enthält Start- und Endzeiten für die instanziierte Prozesskette. Diese referenziert auch alle ProzessschrittLogs welche in dem Auftrag enthalten sind. Aus diesen ist zu entnehmen, wann der einzelne Prozessschritt gestartet worden ist, in welchem Zustand dieser sich befand und wann er beendet worden ist. Falls später dann ein Prozessschritt betrachtet wird, enthält er die Logs und seine ProzessschrittParameter. Die ProzessschrittParameter lassen sich zu JSON serialisieren (Strategie 4.1) 3.2, der ProzessschrittLog sowie der Prozessschritt selber sind auch serialisierbar. Jetzt ist es uns möglich, ein Protokoll der Prozesskette zu serialisieren.

Eine Experimentierstation besteht aus einer eindeutig ID, einem Standort, der als String gespeichert wird, sowie einer Enum, die den Status der Station enthält. Jede der Experimentierstationen enthält eine Warteschlange, die alle Prozessschritt Instanzen enthält, welche auf die Durchführung an dieser Station warten. Um einer Warteschlange zugewiesen zu werden, muss unser System Folgendes tun: Einer ProzessschrittVorlage können Bedingungen hinzugefügt werden, ein Beispiel wäre die Existenz einer Wärmebehandlung. Experimentierstationen können auch Parametern oder Eigenschaften zugewiesen werden. Aus den Bedingungen der ProzessschrittVorlage ergibt sich die Menge an Experimentierstationen, welche möglich sind. Für diese wird geschätzt, wie lange es dauert, bis die bisherige Warteschlange beendet ist. Die Experimentierstation mit der geringsten Wartezeit wird dem Prozessschritt zugeteilt. Dies ist unsere Umsetzung von der Strategie 14.2 3.2.

Zu einer Station können mehrere User vom Administrator zugewiesen werden. User, welche der Station zugeordnet sind, haben die Möglichkeit, den Zustand der Station zu erfragen und zu verändern. Hiermit haben wir die Strategie 6.1 3.2. umgesetzt. Um diese Änderungen zu persistieren, nutzen wir Deltaspike Data. Dafür müssen wir einen EntityManager konfigurieren, der über CDI injected wird. Dies können wir mit dem EntityManagerProducer umsetzen. Für die Experimentierstation erstellen wir nun eine Repositoryklasse. Mit der Query Annotation ist es uns nun möglich, SQL ähnliche Queries für unsere modellierten Klassen zu erstellen.

Jede der Proben enthält eine eindeutige ID, diese entspricht dem ID Schema, welches der Kunde schon in seiner jetzigen Datenbank nutzt. Kommentare und deren Zeitstempel werden als Pair gespeichert (Strategie 21.3) 3.2 und von den Useren bearbeitet. Somit

kann jede Probe in dem System betrachtet werden. Dadurch, dass wir für jede Probe eine ID haben und sie einzeln in der Datenbank speichern, wird Strategie 8.1 3.2 umgesetzt. Somit kann nachvollzogen werden, zu welchem Zeitpunkt sich eine Probe wo und in welcher Prozesskette befunden hat. Weiter kann ein Technologe durch das Kommentieren einzelner Proben wichtige Informationen über eine Probe an seine Mitarbeiter weitergeben. Damit die Nutzer der Probenverwaltung nicht unter langen Ladezeiten leiden, laden wir nur Proben aus der Datenbank, welche wir auch benötigen. Dies wird durch Deltaspike Data und einem Query, der nur einen kleinen Ausschnitt aus der Datenbank lädt, realisiert. Wir nutzen also lazy loading und setzen damit die Strategie 12.1 3.2 um.

Ein Träger enthält Referenzen zu alle Proben, die in ihm enthalten sind. Weiter hat der Träger ein Standort und eine ID. Träger ist eine Assoziationsklasse von Auftrag und Prozessschritt, da manche Prozessschritte nur ausgewählte Trägerarten als Ein- und Ausgabe akzeptieren. Falls sich der Zustand eines Prozessschrittes verändert, und Proben zwischen Experimentierstationen transportiert werden sollen, weisen wir einem Träger, falls nötig, den neuen Schritt zu und setzen alle Proben in den Träger. Das Ziel, der aktuelle Standort, die Proben und deren Träger können dann dem Transport als neuen Transportauftrag zugewiesen werden. Der Transportauftrag ergibt sich impliziert aus dem Auftrag. Er wird dadurch realisiert, dass ein Transporteur nur die für ihn relevanten Informationen erhält. Eine Prozessschrittvorlage hat einen Prozessschrittzustandsautomaten zugewiesen der alle Zustände enthält welche der Prozessschritt anschließend haben kann. Zudem hat jeder Prozessschritt einen Zustandsautomaten, welcher den Zustand des Prozessschritts speichert. Dieser Zustandsautomat kann einen beliebigen Zustand aus dem zur Prozessschrittvorlage zugewiesenem Prozessschritt Zustands Automat Vorlage haben. Grund hierfür ist die Anforderung von dynamischen Zuständen nach Strategie 19.3 3.2. Falls ein Prozessschritt einen Transportauftrag und den Zustand freigeben hat, wird er allen Transporteuren angezeigt. Wenn der Transporteur diesen annimmt, wird der AuftragsZustand auf angenommen gesetzt. Jetzt verschwindet der Auftrag für die anderen Transporteure in der Transportansicht. Der Transporteur läuft nun los und erledigt den Auftrag, danach wird der AuftragsZustand auf abgegeben gesetzt. Der Transporteur erfährt in seiner Ansicht, wo die Proben sich befinden und was ihr nächstes Ziel ist. Er selber wird nicht persistiert. Dies ist unsere Umsetzung von der Strategie 13.1 3.2

. Eine bestimmte Probe kann ausschließlich einem Träger zugeordnet werden.

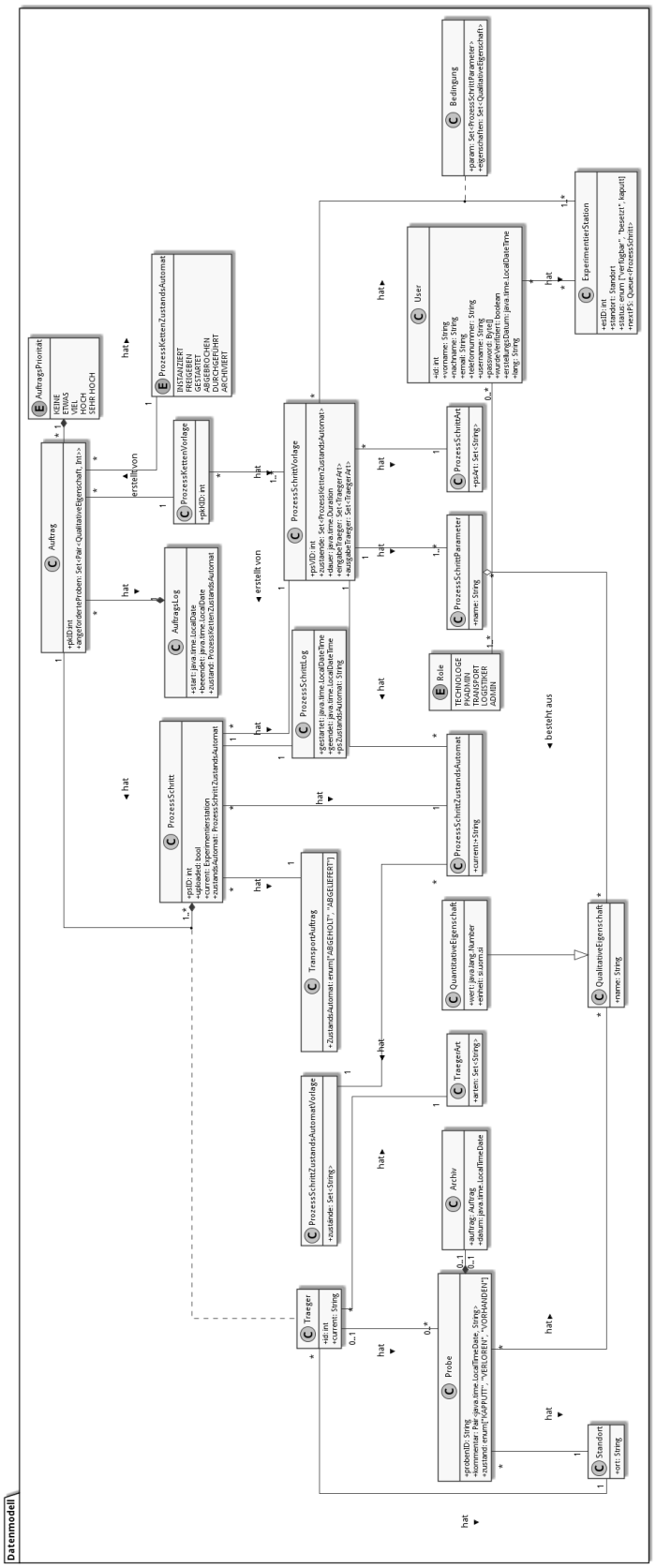

Abbildung 20: Datenmodell für FarbigesSFB

7 Ausführungssicht

Im folgenden Schritt werden wir die Ausführungssicht erläutern. Es wird aufgezeigt, welche verschiedenen Geräte benötigt werden, welche Prozesse auf den Geräten jeweils laufen und welche Module dort enthalten sind.

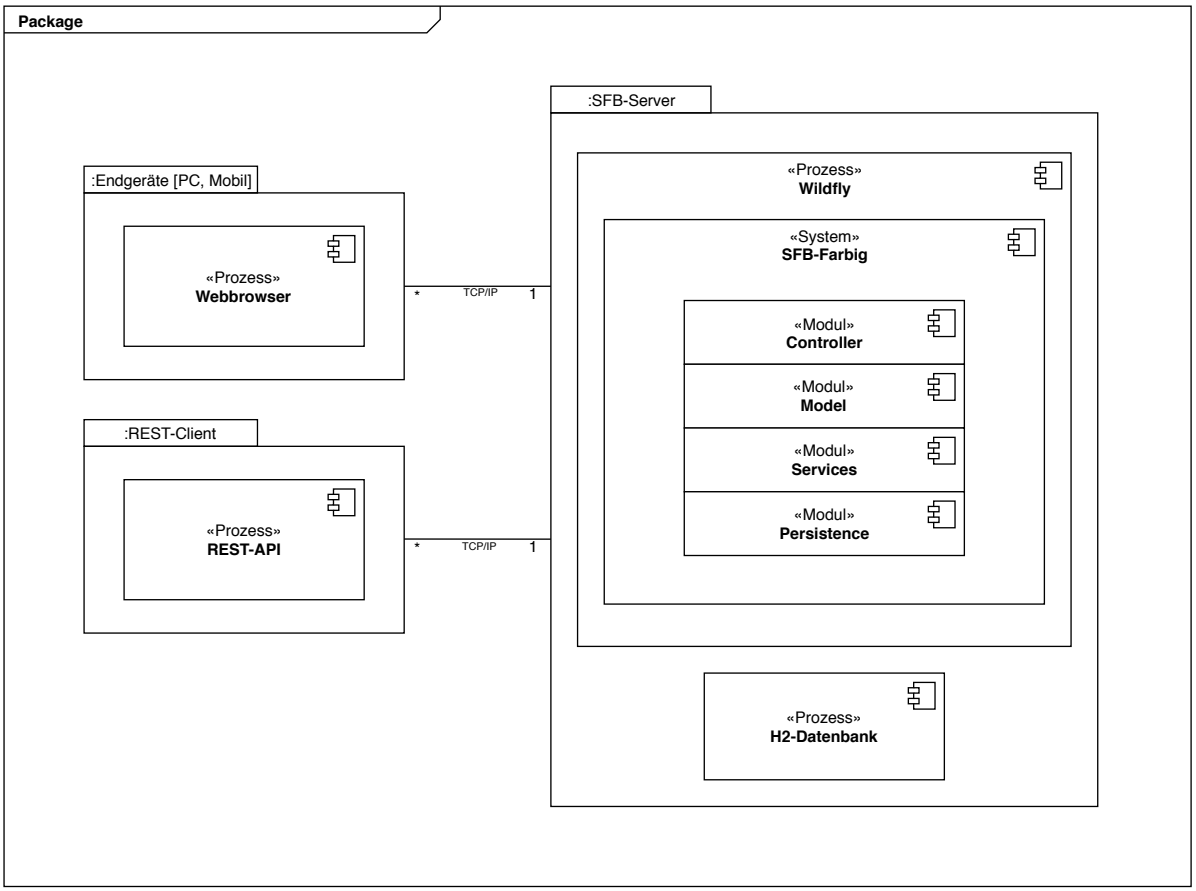

Abbildung 21: Ausführungssicht

Die Ausführungssicht von unserer Software (SFB-Farbig) ist in der obigen Abbildung 7.1 modelliert.

Beliebig viele Benutzer können via TCP/IP auf den SFB-Server zugreifen. Dies geschieht über eine Webapplikation, die im Internetbrowser eines PCs oder eines mobilen Endgeräts aufgerufen werden kann.

Es ist auch möglich, dass ein REST-Client sich mit unserem Server verbindet, dieser verbindet sich dann an den REST-Endpunkt über den SFB-Server. Das Modul der Businesslogik führt die Befehle dann aus.

Es gibt einen Anwendungsserver, welchen wir SFB-Server genannt haben, auf dem eine Instanz von Wildfly läuft. Auf dieser Wildflyinstanz wird unser System (SFB-Farbig)

mit allen dazugehörigen Modulen ausgeführt.

Das Modul Persistence kommuniziert mit dem Modul Businesslogik aber auch „abstract“ über JDBC mit dem H2 Server.

Der SFB-Server stellt eine Verbindung über TCP/IP zu der Datenbank her, welche auf dem gleichen SFB Server läuft und eine Instanz der Datenbanksoftware H2 betreibt.

8 Zusammenhänge zwischen Anwendungsfällen und Architektur

8.1 Login eines Benutzers

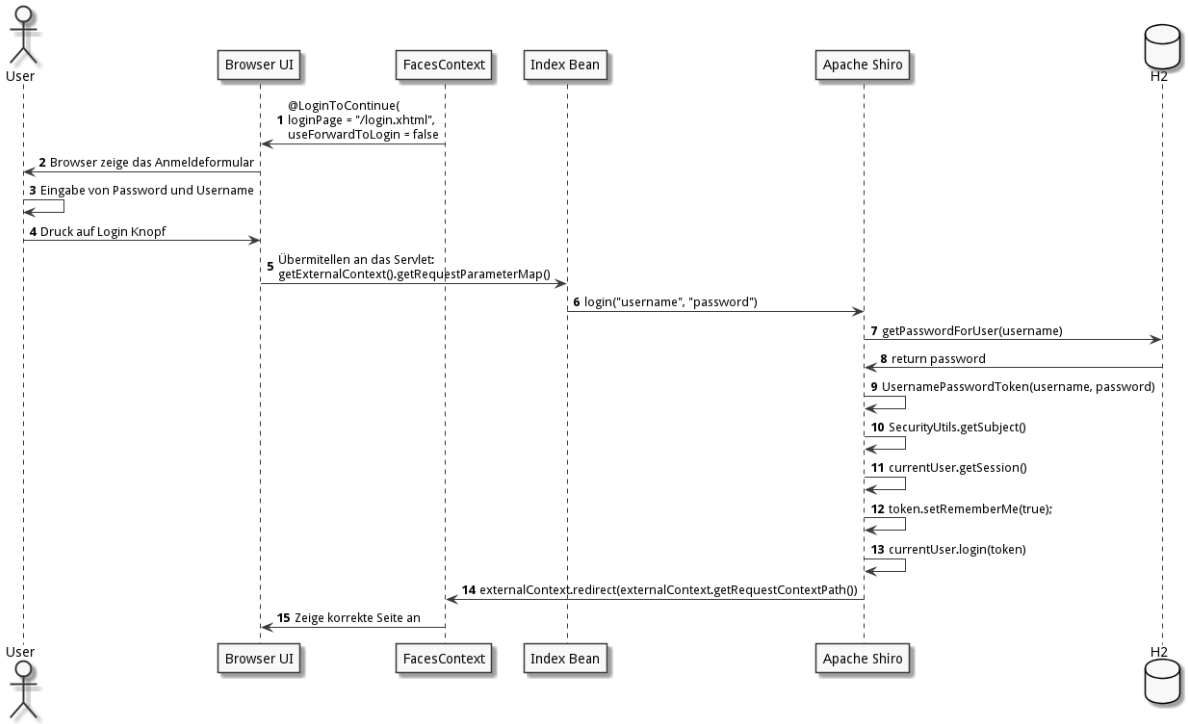

Abbildung 22: Sequenzdiagrammen des Loginvorganges

In diesem Anwendungsfall geht es darum zu zeigen, was passiert wenn sich ein Benutzer anmeldet. Wir haben uns entschieden, die Sicherheit nicht selber zu implementieren, sondern Apache Shiro zu nutzen. Siehe Strategie 2.1 3.2

8.1.1 Was ist Apache Shiro

Als erstes möchten wir einen Überblick über Apache Shiro geben, um dann anhand des Sequenzdiagrammes erläutern zu können, wie Apache Shiro mit unseren Schichten zusammenhängt. Apache Shiro kann ab jetzt auch nur als Shiro bezeichnet werden.

Shiro ist ein sehr mächtiges und leistungsstarkes Java Security Framework, welches sich das Ziel gesetzt hat Entwicklern eine intuitive, jedoch auch umfassende Lösung für Authentifizierung, Autorisierung und Session-Management zu bieten. Shiro basiert auf einem soliden, Schnittstellengesteuerten Design und folgt Prinzipien der Objektorientierung.

Ein *Subject* ist eine sicherheitsspezifische Sicht auf einen Nutzer unseres Programms. Dabei wird nicht zwischen einem menschlichen Nutzer und einem maschinellen unterschieden. Auch in der Sicherheitswelt ist der Begriff Subject eigentlich die anerkannte Nomenklatur. Die Session erlaubt es dem Nutzer, bestimmte Operationen durchzuführen. Es ist weiter möglich einer Session weitere Attribute zu geben. Unsere Session ist eine Shiro-spezifische Instanz, welche das meiste davon bietet, was uns auch eine reguläre HttpSession bietet. Der Vorteil der Session ist es, dass sie keine HTTP-Umgebung braucht.

Bei uns wird die Session in einer Webapplikation umgesetzt, folglich basiert unsere Session auf der standardmäßigen HttpSession.

Wir müssen ein rollenbasiertes Zugriffssystem umsetzen. Siehe Strategie 1.2 3.2. Um dies in Shiro umzusetzen, benutzen wir das von Shiro bereitgestellte JdbcRealm. Mit JdbcRealm kann Shiro aus unserer Datenbank die benötigen Attribute und Rollen eines Nutzers laden.

8.1.2 Erklärung des Diagramms

Es kann vorkommen, dass ein Benutzer unseres Systems in einen Realm navigiert, der nur für bestimmte Rollen zugänglich ist **(Schritt 1)**. Dies ist der Fall für den ersten Schritt unseres Sequenzdiagrammes. Dies passiert in einem FacesContext. Der Webbrowser wird auf die *login.xhml* weitergeleitet. Dort erscheint ein Login Formular **(Schritt 2)**, in dem der Nutzer seine Anmeldedaten eingeben kann **(Schritt 3)**. Das Formular wird vom Browser an unseren Server gesendet **(Schritt 4)**. Für die spätere Verarbeitung brauchen wir die Parameter, die es in dem Kontext der Anmeldung gab. Diese werden in **(Schritt 5)** an das Login Bean gesendet. Das Login Bean hat die Aufgabe, weitere Sicherheitsfunktionen an Shiro zu delegieren. Dafür nutzen die Methode login der Index-Bean. Die hat die Aufgabe, zu ermitteln in welchem Kontext die Login Anfrage gestellt wurde. Sobald dies ermittelt wurde, delegiert sie weiter an Shiro. **(Schritt 6)**. Um das Passwort aus der Datenbank zu lesen, nutzen wir die von Shiro bereitgestellte Methode **(Schritt 7)**. Die Datenbank liefert uns dann das Passwort. In diesem Sequenzdiagramm wurde abstrahiert, dass Passwörter gehashed und gesalted werden. Dies liegt daran, dass Shiro, falls so konfiguriert, dies im Hintergrund macht.

Shiro verlangt ein Token für den Login. Der Token wird aus dem Passwort und dem Benutzernamen generiert. Das genierierte Token hat den Datentyp UserNameToken. Das Token wir in **Schritt 9** generiert.

Ein schon angemeldeter Nutzer kann sich nicht nochmal anmelden. Daher müssen wir überprüfen, ob unser Subject anonym ist **Schritt 10**.

Subjects können Sessions besitzen. Shiro holt und merkt sich die Session eines Subjects in **Schritt 11 & 12**. So haben wir nun ein Subject und seine Session. Hiermit können wir nützliche Dinge tun, beispielsweise überprüfen, ob das Subject die benötigte Rolle hat. Unsere obige Subject Instanz repräsentiert den aktuellen Nutzer, aktuell ist unser Subject aber noch anonym. Um nicht mehr anonym zu sein muss sich ein Subject anmelden. Für den Anmeldeschritt führen wir ein Login mit unserem Token auf unserem aktuellen Nutzer aus **(Schritt 13)**. Nach der erfolgreichem Authentifizierung leiten wir den Nutzer auf den ursprünglichen FacesContext zurück **(Schritt 14)**. Dafür Verwenden wir eine Weiterleitung, die den Pfad zu dem FacesContext kennt, das die Anmeldung gefordert hat.

8.1.3 Kleiner Ausschnitt zur Fehlerbehandlung

Bei dem Login können folgende Exceptions geworfen werden:

- Shiro löst eine UnknownAccountException aus, falls es den Nutzer nicht gibt
- Shiro löst eine IncorrectCredentalException aus, falls das Token nicht stimmt
- Falls das Konto des Users gesperrt ist, wird ein LockedAccountException ausgelöst. Das ist so in der Anwendung bisher nicht vorgesehen, und wird auch vorerst von uns nicht implementiert. Allerdings ist das eine Funktion, die grundsätzlich von Shiro geboten wird und somit auch eventuell von uns implementiert werden könnte.

8.2 Prozessketten Verwalten und Proben beantragen

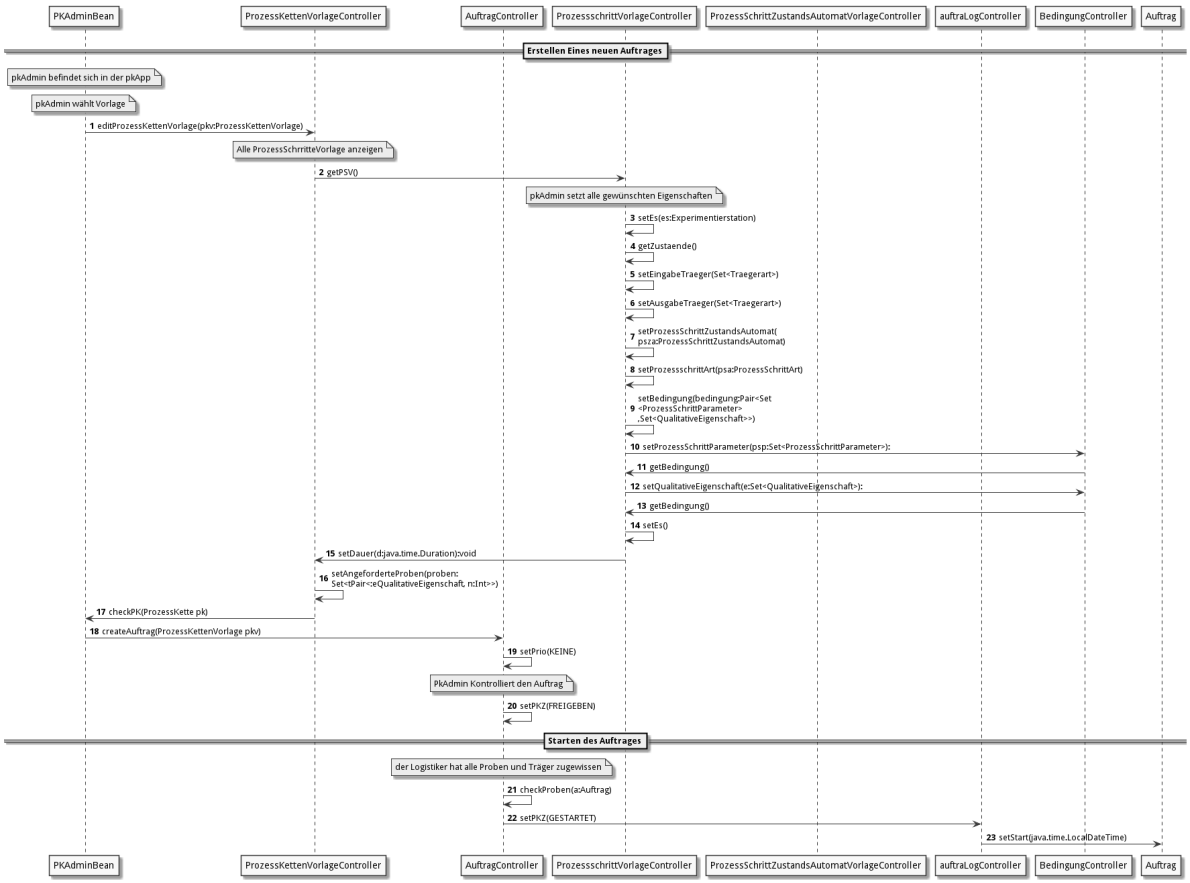

Abbildung 23: Ablauf des Erstellens einer Prozesskette

In der Abb. 23 wird der Ablauf gezeigt, wie ein ProzesskettenAdministrator eine Prozesskette erstellt, diese dann instanziert und dann startet.

Als pkApp App bezeichnen wir den Teil der Webapplikation, der dem Prozesskettenadministrator das Verwalten der Prozessketten ermöglicht.

Sobald der Prozesskettenadministrator der pkApp mitteilt, dass er gerne eine ProzesskettenVorlage bearbeiten würde führt die ProzesskettenadministratorBean die methode $editProzesskettenVorlage(pkv : ProzesskettenVorlage)$ aus (**Schritt 1**). Alls nächstes sollen alle ProzessschrittVorlagen die einer ProzesskettenVorlage zugewiesen sind angezeigt werdne. Dafür führt der ProzesskettenController die Methode $getPSV()$ aus (**Schritt 2**).

Die nächsten Schritte sind optimal. Den es ist dem Prozesskettenadministrator überlassen was er in der Vorlage ändern will. Es bietet sich hier an, manuell eine Experimentierstation zuzuweisen, ansonsten wird dies in dem **11. Schritt** ausgeführt. Es ist dem Prozesskettenadministrator möglich sich den den ProzessschrittZustandsAutomaten des

Prozessschrittes anzeigen zu lassen. Damit die ProzessschrittVorlage sich nachher Experimentierstationen sich zuweisen kann setzt der Prozesskettenadministrator die Eingabe und Ausgabe Träger für die ProzessschrittVorlage. Nun kann der Prozesskettenadministrator die ProzessschrittArt setzten. Anmerkung: Die ProzessschrittArten werden im voraus festgelegt.

Wichtig für die Automatische Zuordnung der ProzessschrittVorlage ist das setzen von Bedingungen. Bedingungen können entweder direkt in der ProzessschrittVorlage generiert werden. Falls eine Bedingung nicht einer trivialen Bedingung entspricht ist es möglich das der BedingungsController diese aus Eigenschaften oder aus ProzessschrittParameteren einem erzeugt. Diese werden dann an den BedingungsController gesendet, dieser gibt dann die korrekte Bedingung zurück. Jetzt kennt der ProzessschrittVorlageController alle Bedingungen für den den Prozessschritt. Die methode setES(), welche keine Parameter hat, ist dafür zuständig aus der Menge a aller Experimentierstationen eine Teilmenge b der Experimentierstationen, die den geforderten Bedingungen entsprechen zu bilden. Alle Experimentierstationen $a \in b$ die der Bedingung entsprechen werden dann der ProzessschrittVorlage zugeordnet

Als letzes setzt der Prozesskettenadministrator die geschätzte Dauer für die Durchführung des Prozessschrittes. Jetzt verwaltet der ProzesskettenVorlageController wieder. Falls der Prozesskettenadministrator jetzt weitere ProzesskettenVorlagen bearbeiten möchte, kann er dies nun tun.

Sobald er die gesamte Prozesskette kontrolliert hat, lässt er die Prozesskette verifizieren. Hier wird überprüft, ob alle Experimentierstationen die benötigen Träger an und ausgeben. Weiter wird in dieser Methode überprüft, ob alle ProzessschrittVorlagen Experimentierstationen zugewiesen bekommen haben. Falls dies nicht klappt, muss der Prozesskettenadministrator die probleme in den ProzessschrittVorlagen selber lösen. Dafür kann er die Randbedingungen des Prozessschrittes ändern oder selber Experimentierstationen der ProzessschrittVorlage zuweisen.

Der Prozesskettenadministrator kann die angeforderten Proben setzen. Sobald die Verifizierung der Prozesskette erfolgt ist, kann er, falls gewollt, die Priorität ändern und dann den Zustand der Prozesskette zu FREIGEGEBEN ändern. Ab diesem Zeitpunkt erscheint in der App der Logistiker die Prozesskette.

Jetzt erhält der Logistiker einen neuen Probenzuweisungsauftrag, falls die Anfrage genehmigt wurde, kann der Prozesskettenadministrator nun die Prozesskette starten.

Hier ist beginnt ein neues Kapitel in unserem Sequenz Diagramm, nämmlich das Starten eines neuen Auftrages. Jetzt erhält der Logistiker einen neuen Probenzuweisungsauftrag, falls die Anfrage genehmigt wurde, kann der Prozesskettenadministrator nun die Prozesskette starten.

Um zu überprüfen ob genügend Proben zugewiesen worden sind, wird die Methode checkProben(a:Auftrag) ausgeführt. Hier wird kontrolliert, ob die Proben die dem Auftrag zugewiesen worden sind den entsprechen die angefordert sind. Jetzt kann mit der Methode setPKZ(GESTARTET) der Auftrag gestartet werden.

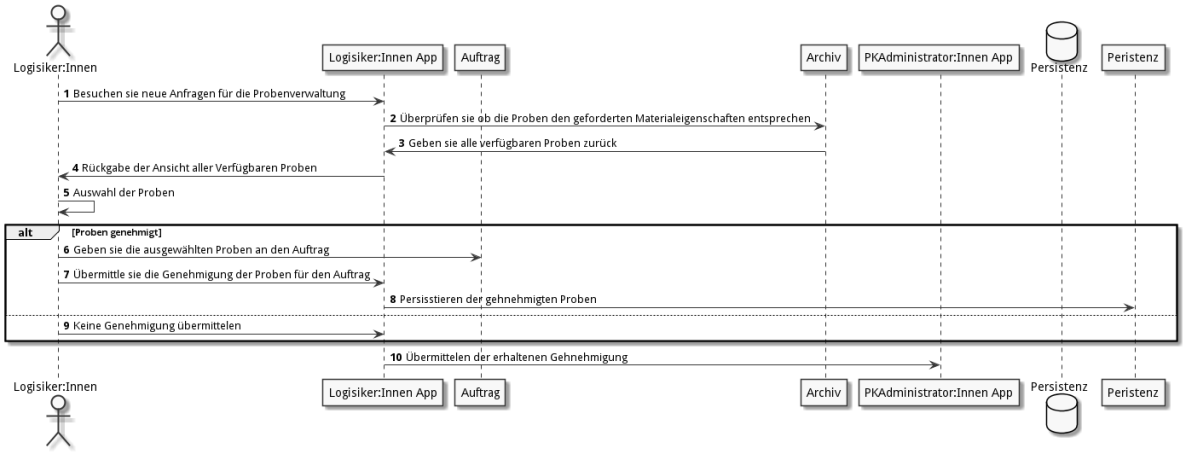

Abbildung 24: Abstrakte Illustration Zuweisen von Proben

In der Graphik 24 erhält die Logistik eine Anfrage für die Probenverwaltung. Es muss nun vom System geprüft werden ob Proben mit den gewünschten Materialeigenschaften im Lager sind. Das System gibt diese an die Logistik zurück. Werden die Proben dann von dem Logistiker ausgewählt und genehmigt, so werden diese an den Auftrag übermittelt und persistiert. Diese Graphik wurde abstrakt gehalten, denn sie Betrifft den Logistiker und nicht den Prozesskettenadministrator

8.3 Prozessschritt Sequenzdiagramm

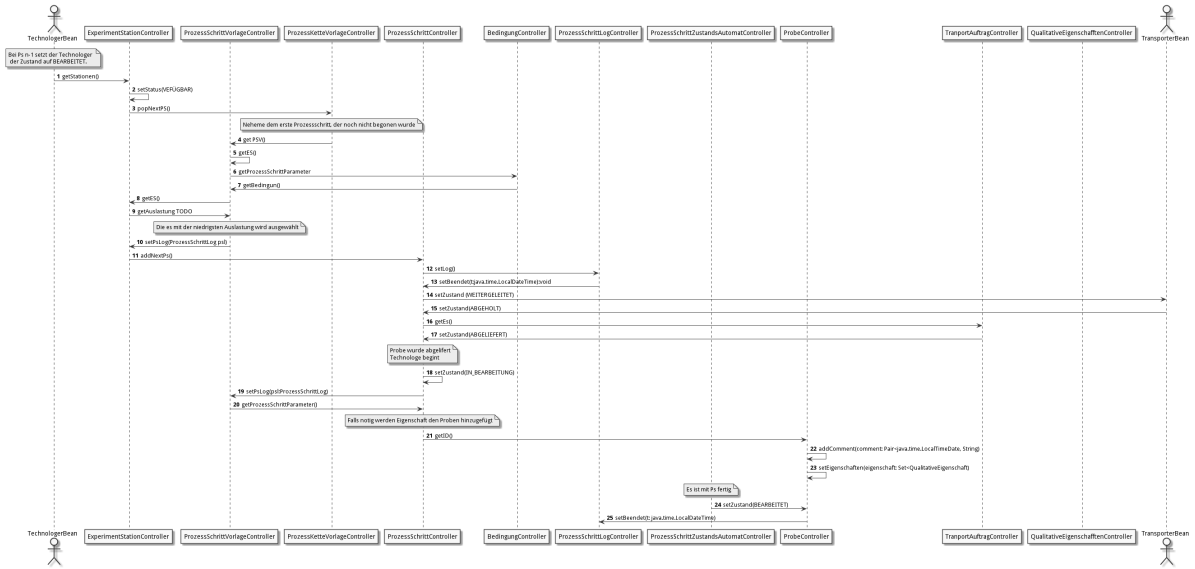

Abbildung 25: Zyklus einer Prozessschrittes

In diesem Anwendungsfall geht es darum, zu zeigen, wie ein Technologe Prozessschritte durchführt. Wir unterscheiden bei Prozessschritten zwischen dem Prozessschritt und der Vorlage. Im Model heißt die Instanz ProzessSchritt und die Vorlage heißt ProzessSchritt-Vorlage hiermit setzen wir die Strategie 3.1 3.2 um. Vorgänge in einem Prozessschritt werden geloggt.

Wir beginnen damit, dass ein Prozessschritt den Zustand BEARBEITET bekommen hat. Dieser Prozessschritt erhält die Bezeichnung $n-1$. Unsere erste Aufgabe ist es, dafür zu sorgen, dass in der Station von $n-1$ der nächste Prozessschritt an die Reihe kommt. Dafür muss als Erster der TechnologenBean getStationen() ausführen. Wir kennen den Prozessschritt, daher wissen wir auch, an welcher Station er ausgeführt wurde. Die Station, an der Prozessschritt $n-1$ sich befindet, erhält den Zustand VERFÜGBAR. Als Nächstes wird der nächste Prozessschritt, der sich in der Warteschlange befindet, aus der Warteschlange entfernt.

Bisher haben wir uns darum gekümmert, dass die Station von dem Prozessschritt $n-1$ bereit ist, mit dem nächsten Prozessschritt zu starten. Nun wollen wir uns darum kümmern, dass der Prozessschritt an die richtige Experimentierstation gelangt. Wir haben uns dafür entschieden, dass dieser Prozess automatisch geschieht. Um dies zu realisieren, nimmt der Prozesskettenvorlagecontroller sich alle Prozesskettenschritte, die in der Prozesskettenvorlage enthalten sind. Hier wird der erste Prozessschritt ausgewählt, der noch nicht begonnen wurde. Als Nächstes brauchen wir die Menge der Experimentierstationen, die dem Prozessschritt zugewiesen worden sind. Auch hier überprüfen wir nochmal ob die Experimentierstationen für den Prozessschritt geeignet sind. Dafür holen wir uns mit dem Prozessschrittvorlagecontroller die Prozessschrittparameter und lassen diese vom Bedingungscontroller als Bedingung zurückgeben. Wir sind uns jetzt sicher, dass die wir die korrekte Menge a der Experimentierstationen haben an denen wir den Prozessschritt n zuweisen können. Als Nächstes wollen wir wissen, welche der Experimentierstationen aus der Menge n die am wenigsten ausgelastete ist. Der ExperimentierStationenController übernimmt für uns diese Aufgabe, dies wird mit getAuslastung realisiert. Die Auslastung ist eine Zeitdauer, aus der Menge der Auslastungen nehmen wir die Experimentierstation mit der geringsten Auslastung. Wir schreiben die Zuweisung in das Log des Prozessschrittes. Nun fügen wir den Prozessschritt n in die Warteschlange der vorher ausgewählten Experimentierstation. Die Zuweisung an eine spezifische Experimentierstation wird auch in dem Log von n durch den ProzessSchrittLogController vermerkt. Der Prozessschritt $n-1$ ist nun offiziell beendet, daher setzen wir den Zustand vom Prozessschritt $n-1$ auf WEITERGLEITET.

Jetzt geht es darum, dass der Transporteur erkennt, dass er einen neuen Transportauftrag hat, um dies zu realisieren können wir schauen, ob es einen Prozessschritt n gibt, der noch nicht begonnen wurde, aber schon zu einer Experimentierstation zugewiesen wurde. Sein Vorgänger $n-1$ muss außerdem den Zustand WEITERGLEITET haben. Wenn dies gilt, sehen alle Benutzer mit der Rolle Transport den Transportauftrag für die Probe n. Sobald der die Probe abgeholt wurde, wird der Zustand des Transportauftrages auf ABGEHOLT gesetzt. Sobald der Transporteur an der Experimentierstation angelangt

ist wo Prozessschritt n ausgeführt wird. Wird der Transportauftrag auf ABGELIEFERT gesetzt.

Ab diesem Zeitpunkt beginnt der Technologe mit der Durchführung vom Prozessschritt n. Als Erstes setzt er den Zustand von Prozessschritt n auf IN_BEARBEITUNG. Diese Zustandsänderung wird auch sofort im Log des Prozessschrittes n vermerkt. Um zu wissen, was zu tun ist, werden die Prozessparameter aus der Prozessschrittvorlage geladen. Es ist möglich, dass die Proben neue Eigenschaften durch den Prozessschritt erhalten haben. Falls dies der Fall ist, müssen alle Proben die in dem Träger waren, die vom Transport geliefert werden die neuen Eigenschaften erhalten. Hier ist es auch für den Technologen möglich Kommentare an die einzelnen Proben hinzuzuführen. Sobald alles protokolliert wurde, setzt der Technologe den Zustand des Prozessschrittes auf BEARBEITET. Nun erfolgt das das erstellen des Logs für den Prozessschritt. Jetzt kann ein weiter Prozessschritt erfolgen.

9 Evolution

Um die Anwendung nach der Auslieferung an den Kunden entwickel- und erweiterbar zu halten, wird die Software in verschiedene Module aufgeteilt und eine gute Entkopplung & Kapselung dieser kommen der Erweiterung der Software zugute. Im Folgenden präsentieren wir Ideen zur Erweiterung der Software.

9.1 Automatische statistische Auswertung

Die erste Idee zum sinnvollen Erweitern der Software soll eine Voraussicht ausgeben können, wie sich nach dem Ausführen eines beliebigen Prozessschritts auf eine Probe ihre Eigenschaften ändern würden, bzw. welche Eigenschaften die Probe danach wahrscheinlich haben wird. Da wir in der Datenbank ca. eine halbe Million Proben haben, kann ein analytischer Algorithmus Statistiken erstellen, welche die höchste Wahrscheinlichkeit einer Veränderung der Eigenschaften einer Probe auswertet. Um es in unsere Architektur einzubinden, muss eine Klasse für die Statistiksoftware erstellt werden, aus dieser werden die Statistiken erstellt und dann mit den eingegebenen Daten abgeglichen. Am Ende bekommt man die Änderung der Eigenschaften zurück, die am wahrscheinlichsten ist, wenn ein Prozessschritt auf die Probe angewandt wird.

9.2 Voraussichtliche Dauer eines Prozesses

Man kann den Algorithmus aus der automatischen statistischen Auswertung auch dafür verwenden, dass aus den durch den Algorithmus aus 9.1 errechneten empirischen Daten die voraussichtliche Dauer eines Prozesses errechnet und ausgegeben wird. Hier braucht man, wie in 9.1, eine neue Klasse, welche die Statistiksoftware beinhaltet, aus welcher man dann eine wahrscheinliche Zeit herausbekommt.

9.3 Scanner

Da so gut wie jedes mobile Endgerät eine Kamera besitzt, könnte man diese nutzen, um QR-Codes, die auf den Proben, den Trägern, im Archiv und an jeder Station zu finden sind, zu scannen damit z. B. der Transporter nicht mehr manuell eingeben muss, welche Datei er wohin bringt.
Er scannt den QR-Code des Trägers und dann die QR-Codes beliebig vieler Proben. Die Software ordnet nun automatisch die gescannten Proben dem gescannten Träger zu. Er kann den QR-Code des Trägers scannen und die Software weiß automatisch, dass der angemeldete Transporter den Träger abgeholt hat und wenn er jetzt den QR-Code der nächsten Experimentierstation scannt, trägt die Software automatisch den Träger als abgeliefert ein.

Um dies zu realisieren, muss die REST-API so erweitert werden, dass über die Businesslogik die Proben und Träger im Modell aktualisiert werden. Der Zustand der jeweiligen Prozessschritte muss ebenfalls verändert werden.