

# **TR-Manager**

**Softwaregestützte Verwaltung des Trainingsraumkonzeptes an Schulen.**

**Große Studienarbeit**

**von**

**Alexandra Neffgen & Florian Baier**

**Juni 2016**

Kurs: TINF13ITNS

DHBW-Betreuer: Dr. Rainer Colgen

Bearbeitungszeitraum: 19.06.2015

Alexandra Neffgen 4788873

Florian Baier 8514884

## EIDESSTATTLICHE ERKLÄRUNG

---

### Eidesstattliche Erklärung

gemäß §5 (2) der “Studien- und Prüfungsordnung DHBW Technik” vom 18. Mai 2009.

Hiermit versichern wir, dass wir unsere Praxisarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

---

Ort, Datum

---

Alexandra Neffgen

---

Florian Baier

### Zusammenfassung

Störungen des Unterrichts durch Schüler zerren stets sowohl an den Nerven des Lehrer, als auch an denen der Mitschüler. Eine ausreichende Besprechung und Reflexion dieser Störung während des Unterrichtes kostet allen beteiligten viel Zeit, die dem Unterricht folglich fehlt. An dieser Stelle greift das Trainingsraumkonzept, in dem es die Nachbereitung der Störung mit dem Schüler aus dem Unterricht in einen anderen betreuten Raum verlegt. Für die Organisation rund um den Trainingsraum gibt es bisher keinerlei softwaregestützte Hilfe, daher beruht die bisherige Verwaltung auf Papier.

Im Rahmen der vorliegenden Studienarbeit wird eine Anwendung entwickelt, welche die Lehrer unterstützen und vieles der Papier gestützten Verwaltung ersetzt. Einleitend wird zunächst die Idee und die Vorgänge des Trainingsraum im Detail erklärt. Folgen wird die konzeptionelle Entwicklung der Anwendung so wie die technischen Grundlagen für das Verständnis der Umsetzung. Die Anwendung lässt sich in mehrere Teile separieren, deren Umsetzung ebenfalls in verschiedenen Kapitel gegliedert ist.

Das Fazit widmet sich den unterschiedlichen Funktionen und der Vollständigkeit ihrer Implementierung. Im Ausblick wird das weitere Potenzial der Anwendung erfasst und die Bestrebungen der Autoren und Entwicklern beleuchtet. Während des gesamten Prozesse stand den Entwicklern das Robert-Wetzlar-Berufskolleg Bonn zu Seite. Dieses lieferte zum einen die Anforderungen und Testdaten und zum anderen wurde dort im laufenden Betrieb die Anwendung getestet und evaluiert.

## Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>I</b>
<b>Quellcodeverzeichnis</b>	<b>II</b>
<b>Abkürzungsverzeichnis</b>	<b>III</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 These und Entwurf . . . . .	2
1.3 Ziele und Aufgabenstellung . . . . .	2
1.4 Stand der Technik . . . . .	3
1.4.1 Aktuelle Unterstützungen . . . . .	3
1.4.2 Rechtliches . . . . .	3
1.5 Aufbau der Arbeit . . . . .	3
<b>2 Konzept</b>	<b>5</b>
2.1 Trainingsraum . . . . .	5
2.2 User-Stories . . . . .	6
2.3 Usability . . . . .	8
2.4 Design . . . . .	9
2.5 Backend . . . . .	9
2.6 Datenbank . . . . .	10
2.7 Clients . . . . .	10
2.7.1 Native . . . . .	10
2.7.2 Mobile . . . . .	11

## INHALTSVERZEICHNIS

---

2.7.3	Web . . . . .	11
2.8	VPN . . . . .	12
<b>3</b>	<b>Grundlagen</b>	<b>13</b>
3.1	Datenbanken . . . . .	13
3.2	REST . . . . .	14
3.3	Java . . . . .	15
3.3.1	JUnit . . . . .	16
3.4	Spring . . . . .	16
3.4.1	Spring Boot . . . . .	17
3.4.2	Spring MVC . . . . .	17
3.4.3	Spring Data JPA . . . . .	17
3.4.4	Dependency Injection . . . . .	18
3.4.5	Aspect Oriented Programming . . . . .	18
3.5	.NET Framework . . . . .	19
3.5.1	Windows Forms . . . . .	19
3.5.2	ASP.NET . . . . .	20
3.5.3	Model View Controller (MVC) . . . . .	20
3.6	Logischer Aufbau einer Anwendung . . . . .	22
3.7	Web-Applikationen . . . . .	23
3.8	Frameworks . . . . .	24
3.8.1	Bootstrap . . . . .	24
3.8.2	Zend . . . . .	25
3.8.3	Java Server Faces . . . . .	25
<b>4</b>	<b>Umsetzung des Backends</b>	<b>26</b>
4.1	Anforderungen . . . . .	26

## INHALTSVERZEICHNIS

---

4.2	Implementierung . . . . .	26
4.2.1	Model . . . . .	27
4.2.2	Repositories (Spring Data) . . . . .	29
4.2.3	Services . . . . .	30
4.2.4	Controller . . . . .	32
4.3	Tests . . . . .	34
4.3.1	Controller-Tests . . . . .	34
<b>5</b>	<b>Clients</b>	<b>40</b>
5.1	Anforderungen . . . . .	40
5.2	Implementierung . . . . .	40
5.2.1	Model . . . . .	41
5.2.2	HTTP-Client . . . . .	43
5.3	Nativer Client . . . . .	48
5.3.1	Design . . . . .	48
5.3.2	Workflows . . . . .	49
5.4	Web Client . . . . .	57
5.4.1	Funktionsumfang . . . . .	57
5.4.2	Design . . . . .	57
<b>6</b>	<b>Fazit</b>	<b>59</b>
<b>7</b>	<b>Ausblick</b>	<b>61</b>
7.1	Technische Komponenten . . . . .	61
7.1.1	Web-Applikation . . . . .	61
7.1.2	Mobile Applikation . . . . .	61
7.1.3	Statistik-Tools . . . . .	62

## INHALTSVERZEICHNIS

---

7.1.4	Backend-Erweiterungen . . . . .	62
7.2	Wirtschaftlicher Faktor . . . . .	62
7.2.1	Marketing . . . . .	63
7.2.2	Lizensierung . . . . .	63
<b>8</b>	<b>Glossar</b>	<b>66</b>
<b>9</b>	<b>Anhang</b>	<b>68</b>
9.1	LazyObject . . . . .	68
9.2	refreshInterfaceData . . . . .	71
9.3	configurationStore . . . . .	73
9.4	User-Stories . . . . .	77
9.5	Anwenderhandbuch . . . . .	82

## Abbildungsverzeichnis

1	Interaktionen im MVC-Modell . . . . .	21
2	Schichtenarchitektur einer Webanwendung . . . . .	23
3	ER-Modell der Datenbank . . . . .	28
4	Design des Webclients . . . . .	58



## Quellcodeverzeichnis

1	Definition eines Repositories . . . . .	30
2	Definition eines Services . . . . .	31
3	Definition eines Controllers . . . . .	33
4	Test der getById-Methode des Teacher-Controllers . . . . .	36
5	Test der Add-Methode des Teacher-Controllers . . . . .	37
6	testAuthenticate des User-Controllers . . . . .	38
7	Teacher-Model-Klasse im Client . . . . .	42
8	getById-Methode des Http-Clients . . . . .	43
9	Add-Methode des Http-Clients . . . . .	44
10	Klasse ausgewählt . . . . .	49
11	Speicherung der Konfiguration . . . . .	54
	listings/LazyObject.java . . . . .	68
	listings/refreshInterfaceData.cs . . . . .	71
	listings/configuration_store.cs . . . . .	73

# Abkürzungsverzeichnis

<b>REST</b> Representational State Tranfer .....	9
<b>HTTP</b> Hyper Text Transfer Protocol .....	10
<b>HTTPS</b> Hyper Text Transfer Protocol Secure .....	26
<b>LAN</b> Local Area Network .....	2
<b>AJAX</b> Asynchronous Javascript and XML .....	11
<b>ASP</b> Active Server Pages .....	11
<b>JSF</b> Java Server Faces .....	11
<b>ORM</b> Object Relational Mapping .....	
<b>DRY</b> Don't repeat yourself .....	24
<b>JDBC</b> Java Database Connectivity .....	16

<b>JPA</b> Java Persistence API.....	16
<b>JMS</b> Java Message Service .....	16
<b>ASP</b> Active Server Pages.....	11
<b>MSIL</b> Microsoft Intermediate Language.....	20
<b>IoC</b> Inversion of Control .....	18
<b>AOP</b> Aspect Oriented Programming	
<b>WWW</b> World Wide Web .....	12
<b>VPN</b> Virtual Private Network .....	12
<b>OSS</b> Open Source Software.....	10
<b>DBMS</b> Datenbankmanagement-System .....	13
<b>URI</b> Uniform Resource Identifier .....	32

<b>MVC</b> Model View Controller .....	48
<b>FK</b> Foreign Key .....	29
<b>ODBC</b> Open Database Connectivity .....	13
<b>TLS</b> Transport Layer Security .....	15
<b>API</b> Application Programming Interface .....	19

# 1 Einleitung

Im Rahmen dieser Arbeit wird eine Anwendung entwickelt, welche Pädagogen an Schulen eine effiziente Umsetzung des Trainingsraum-Konzeptes ermöglicht. Dieses Kapitel erläutert die Motivation hierfür, erarbeitet eine These und definiert auf dieser Basis eine Aufgabenstellung und Ziele. Ferner wird der aktuelle Stand der Technik vorgestellt und einen Ausblick auf die Kapitel dieser Arbeit gegeben.

## 1.1 Motivation

In vielen Schulklassen sämtlicher Schulformen kommt es immer wieder zu Störungen des Unterrichtes durch Schüler. Dadurch verliert der Unterricht inhaltlich nicht nur an Qualität, sondern auch an Schwung. Um dem entgegen zu wirken werden an vielen Schulen die Abwicklung der Unterrichtsstörung und der Unterricht selber von einander getrennt. So werden die störenden Schüler aus dem Unterricht heraus genommen und erhalten in einem so genannten Trainingsraum Unterstützung, um die Störung aufzuarbeiten und sich sozial weiterzuentwickeln. Dieses Konzept wurde erstmals von Edward E. Ford in Phoenix, Arizona eingesetzt. Hier in Deutschland ist es daher sowohl unter dem Namen Trainingsraumprogramm, als auch Arizona-Modell bekannt.

In Zusammenhang mit einem Besuch im Trainingsraum sind einige formale Dinge zu beachten. Gerade bei mehrfachem Besuch im Trainingsraum müssen neben dem Standardvorgehen weitere Prozesse angestoßen werden. Dies mit rein analogen Mitteln oder einfachen digitalen Ressourcen wie etwa Excel umzusetzen, ist nicht nur aufwändig, sondern auch anfällig für Fehler und schwer zu überblicken.

An dieser Stelle soll der Trainingsraum Manager Anwendung finden. Er soll nicht nur als Verwaltungstool, sondern auch den betreuenden Lehrern als Orientierung für den Ablauf dienen. Dies soll obendrein die Einarbeitung neuer Lehrkräfte in das Konzept erleichtern.

## **1.2 These und Entwurf**

Das Projekt TRManager soll es den Pädagogen ermöglichen sich ganz auf die Förderung der Schüler zu konzentrieren. Durch eine grafische Benutzeroberfläche werden die Pädagogen durch das Programm geführt. Sie erhalten eine Übersicht der möglichen Aktionen, sowie eine Übersicht der anwesenden Schüler. Außerdem werden die Regeln und Vorgaben, wie etwa die Mindestanwesenheits-Dauer eines Schülers, implementiert. Dies ermöglicht einen einheitlichen Ablauf. Sämtliche Daten werden in einer Datenbank noch während der Laufzeit gespeichert. Zum einen lässt sich dadurch die Datenkonsistenz sicherstellen, so dass bei einem unerwartendem Beenden der Anwendung auch die aktuellen Daten in der Datenbank zu finden sind. Nach einem Neustart der Anwendung lässt sich also direkt weiter arbeiten. Außerdem ist so möglich den aktuellen Stand des Trainingsraumes von mehreren Clients aus einzusehen. Grundsätzlich soll eine Analyse der Daten mit Hilfe von Statistiken möglich sein. Diese sollen nicht nur der Analyse der Schüler dienen, sondern auch der Pädagogen selbst.

## **1.3 Ziele und Aufgabenstellung**

Ziel dieser Arbeit ist es, eine Server-Client-basierte Applikation zu erstellen. Das Backend (der Server) soll auf einer Maschine in der jeweiligen Schule laufen. Der Client wird sowohl auf der gleichen Maschinen laufen können, als auch auf jeder anderen Maschine, die sich im gleichen Local Area Network (LAN) befindet. Es ist sowohl das Backend zu programmieren, als auch ein entsprechender Client. Des Weiteren muss ein geeignetes Protokoll zu deren Kommunikation gewählt werden. Am Ende der Arbeit soll eine Anwendung als Prototyp stehen.

## **1.4 Stand der Technik**

Der Stand der Technik beschreibt das technische Umfeld das zu dieser Technik besteht. Es werden die bestehenden Konzepte sowie die rechtliche Situation erläutert.

### **1.4.1 Aktuelle Unterstützungen**

Das Trainingsraum-Konzept wird bereits in vielen Schulen verwendet. Dabei meist an weiterführenden Schulen und eher selten an Berufkollegs oder Grundschulen. Sowohl auf der Webseite des Landes Baden-Württemberg, als auch auf den dort verlinkten Seiten existieren Materialien zur Unterstützung. Eine digitale Unterstützung in Form eines Programmes ist allerdings bisher nicht zu finden. Diese Lücke soll mit dieser Studienarbeit gefüllt werden.

### **1.4.2 Rechtliches**

Das Modell als solches stammt aus Amerika und wird unter anderem vom Land Baden-Württemberg publiziert. Zwar sind auf dem deutschen Markt Bücher zu der Thematik veröffentlicht, aber auf dem Konzept als Solches bestehen keine Rechte. Da in der Studienarbeit selber auch keine Materialien verwendet werden, welche unter Copyright stehen, kann hier völlig frei gearbeitet werden. Sämtliche verwendete Unterlagen und Beispieldaten wurden vom Robert-Wetzlar-Berufskolleg Bonn zur Verfügung gestellt.

## **1.5 Aufbau der Arbeit**

Diese Arbeit umfasst den Entwicklungsprozess vom Konzept bis hin zu einem Prototypen. Dabei wird unter anderem die Entscheidungsfindung für unterschiedliche Technologien sowie die Technologien selbst genauer betrachtet. Des Weiteren werden das Zusammenspiel der verwendeten Technologien erläutert.

Kapitel Zwei umfasst den konzeptionellen Aufbau des Trainingsraum-Manager. Es wird dabei auf die unterschiedlichen Aspekte der Implementierung eingegangen und konzeptionelle Rahmenbedingungen betrachtet.

Kapitel Drei enthält die technischen Grundlagen mit welchen das Programm umgesetzt ist.

In Kapitel Vier wird die Implementierung der Serveranwendung besprochen. Diese ist für die Verwaltung der Daten verantwortlich

Kapitel Fünf hingegen umfasst die Umsetzung der Clients. Diese präsentieren die Daten und stellen außerdem grafische Benutzeroberflächen dar.

Die abschließenden Kapitel sechs und sieben geben ein Fazit über das Projekt in Bezug auf die Anforderungen, sowie einen Ausblick welches weitere Potenzial die Anwendung birgt.



## 2 Konzept

In diesem Kapitel wird der konzeptuelle Aufbau des Projektes beschrieben. Dabei werden zuerst die abstrakten Konzepte, dann die Anforderungen des Kunden beziehungsweise die User-Stories und zum Schluss Konzepte zur technischen Umsetzung dargestellt.

### 2.1 Trainingsraum

Das Konzept des Trainingsraumes beruht im wesentlichen auf den folgenden drei Regeln:

- Jeder Schüler/jede Schülerin hat das Recht ungestört zu lernen.
- Jeder Lehrer/jede Lehrerin hat das Recht ungestört zu unterrichten
- Jeder muss die Rechte der Anderen respektieren

Diesen drei Regeln, oder auch drei Rechte genannt, stimmt jeder Schüler und Lehrer zu. Sollte ein Schüler wiederholt den Unterricht stören, wird dieser ausdrücklich ermahnt und vor die Wahl gestellt entweder die Regeln einzuhalten und das Recht der anderen zu beachten oder aber in den Trainingsraum zu gehen. Dadurch wird der Aufwand der zur Aufarbeitung der Störung benötigt wird, aus dem Unterricht herausgenommen werden und die interessierten Schüler können effektiver arbeiten. Den störenden Schülern wird es so auch ermöglicht in einer entspannten Atmosphäre ihr Verhalten und die Folgen für sich selbst und andere zu reflektieren. Im Trainingsraum können sie eigenständig an Ideen zu Verbesserung ihres Verhalten arbeiten und einen konkreten Plan ausarbeiten, um an der Verbesserung zu arbeiten. Zur Unterstützung ist im Trainingsraum während der Unterrichtszeiten stets ein Pädagoge anwesend, welcher vorwurfsfrei die Schüler bei ihren Ausarbeitungen unterstützt und Hilfestellungen bietet.

## 2.2 User-Stories

User Stories sind in Alltagssprache definierte Software-Anforderungen. Dabei umfassen solche Anforderungen in der Regel 2-3 Sätze. In Zusammenarbeit mit einem Berufskolleg, welches bereits seit längerer Zeit mit dem Trainingsraumkonzept arbeitet, wurden entsprechende Anforderungen formuliert. Für eine bessere Übersicht wurden die einzelnen Punkte zum einem nach Grundfunktionen und unterstützende Funktionen aufgeteilt und zum anderen in logische Gruppen zusammengefasst. Die ausführlichen User-Stories sind im Anhang 9.4 auf Seite 77 zu finden. Folgend werden die wesentlichen Gruppen erläutert.

- Import und Export der Daten / Datensicherung (Priorität 1)
  - Um mit den aktuellen Daten der jeweiligen Schule arbeiten zu können, werden diese mittels CSV-Dateien importiert und können zur Absicherung auch wieder in CSV-Dateien exportiert werden
- Login Daten (Priorität 1)
  - Jeder Lehrer kann sich durch dem ihm zugeordneten Kürzel ohne Passwort anmelden. Die administrativen Funktionen sollen Passwortgeschützt sein.
- Besuch eines Schülers eintragen (Priorität 1)
  - Die Haupttätigkeit eines Lehrer im Trainingsraum
- Vervollständigung (Priorität 1)
  - Zur Vereinfachung der Namensfindung von Lehrern, Schülern und Klassen soll es eine Art Autovervollständigung geben.
- Behandlungen bei drei bzw. fünf Besuchen (Priorität 1)

- Nach drei bzw. fünf Besuchen eines Schülers im Trainingsraum sind von schulischer Seite besondere Maßnahmen zu treffen. Auf diese soll durch das Programm hingewiesen werden.
- Behandlung Rückkehrer TR aufgrund von Ablehnung Rückkehrplan (Priorität 1)
  - Rückkehrer werden auch technisch anders als ein Erstbesuch im Trainingsraum behandelt, da die Scheinnummer die gleiche bleiben muss.
- Administrationsrechte (Priorität 1)
  - Sämtliche verwaltende und zeitunabhängige Datenmanipulation geschieht über einen Passwortgesicherten Account.
- Schülerdatenabruf / Information über Besuche im Trainingsraum (TR) (Priorität 2)
  - Statistische Informationen zu einem Schüler sollen für den administrativen Account abrufbar sein.
- Zeitzähler (Priorität 2)
  - Ein Schüler muss mindestens 15 Minuten im Trainingsraum sich seinem Rückkehrplan widmen. Diese 15 Minuten sollen dem betreuenden Lehrer grafisch angezeigt werden
- Druckfertige Nachricht (Priorität 4)
  - Der Hinweis für die Schule zu einem dritten bzw. fünften Besuch soll als druckfertige Nachricht ausgegeben werden
- Lehrer mit hinterlegter Mailadresse (Priorität 4)

- Zur Modernisierung der Benachrichtigungen an die Lehrer, soll deren Mailadresse hinterlegt werden
- Mailfunktion (Priorität 5)
  - Sämtliche relevanten Informationen an die Lehrer sollen zeitnah per Mail verteilt werden
- Statistiken anhand der Exportdaten (Priorität 4)
  - Anhand der exportierten Daten sollen lokal Statistiken erstellt werden können (Filter- und Sortieroptionen).
- Externer Zugriff (Priorität 5)
  - Für die Administration soll ein externer Zugriff auf die Daten über eine zusätzliche Software Komponente möglich sein.

## 2.3 Usability

Zu Usability gibt es keine einheitliche Definition, da Usability in vielen Bereichen unseres Lebens wichtig ist. So gibt es das Thema Usability bei der Nutzung von Software, aber auch Verpackungen oder Prozesse müssen „usable“ sein. In der DIN EN ISO 9241-11 wird Usability definiert als ein Maß dafür, wie gut ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend (aus arbeitspsychologischer und -physiologischer Sicht) zu erreichen. Diese Definition ist in ihrem Kern weitestgehend akzeptiert, jedoch wird von einigen Autoren dieser Begriff noch erweitert um Elemente wie Spaß oder eine Möglichkeit der Mitgestaltung des Systems. Steve Krug verfolgt dabei einen deutlich simpleren Ansatz mit seiner Definition von Usability: „Eine Person

von durchschnittlicher (oder niederer) Begabung und Erfahrung kann sich erschließen, wie sie etwas zum Umsetzen ihrer Aufgabe benutzen muss, ohne dabei mehr Schwierigkeiten zu haben als es ihr wert ist“<sup>1</sup>. Krug unterstützt deshalb auch nicht die Regeln, die beispielsweise besagen, dass jede Seite nur maximal fünf Klicks entfernt sein darf. Er führt an, das auch mehr als fünf Klicks in Ordnung sind, solange ein Klick ohne nachdenken passiert und dem Nutzer ´anzeigt, dass er auf dem richtigen Weg befindet. Die Definition von Krug soll in dieser Arbeit als die Basis genommen werden.

## 2.4 Design

Die Applikation besteht im Grunde aus drei Teilen: Dem Frontend, dem Backend und der Datenbank. Die Datenbank speichert die Daten, die erfasst werden und präsentiert diese dem Backend. Das Backend verarbeitet die Daten, die es vom Frontend und Datenbank bekommt, um es an beide weiterzureichen. Das Frontend interpretiert die Daten, die es vom Backend geliefert bekommt und zeigt diese dem Nutzer an und nimmt Eingaben vom Nutzer entgegen und schickt diese an das Backend.

## 2.5 Backend

Das Backend dient als Schnittstelle zwischen Datenbank und Frontend und bereitet die Daten, die von Datenbank und Frontend kommen, für den jeweils anderen auf. Umgesetzt wird das Backend mit Spring. Die Zugriffe erfolgen über einen RESTful-Service, der laut REST-Spezifikation die normalen HTTP-Methoden verwendet (GET, POST etc.). Die Verwendung von Representational State Transfer (REST) ermöglicht es, die Clients unabhängig vom Server zu entwerfen, da die Methoden standardisiert

---

<sup>1</sup> Steve Krug. *Dont make me think, revisited: a common sense approach to Web usability*. New Riders, 2014, Seite 9

und für jeden Client gleich sind. Auch andere Applikationen können so ohne Probleme auf diesen Dienst zugreifen, um zum Beispiel die gesammelten Daten anderweitig zu verwenden.

## 2.6 Datenbank

Die Datenbank-Struktur wird aus den annotierten Objektklassen generiert. Ermöglicht wird dies durch Hibernate, dies wird im Kapitel 4.2.1 auf Seite 28 erläutert. Als Datenbankserver soll MySQL zur Verwendung kommen, da dies für die relativ geringe Datenmenge und die clientseitige Backupstrategie im vorliegenden Kontext ausreichend sind. Zudem fallen bei MySQL keine Lizenzierungskosten an, da die Community-Version als Open Source Software (OSS) zur Verfügung steht.

## 2.7 Clients

Die Clients haben den Zweck, dem Benutzer die gewünschten Daten anzuzeigen. Der Zugriff auf diese Daten erfolgt generisch über Hyper Text Transfer Protocol (HTTP)-Requests(REST), so dass für jeden Client die gleiche Technik verwendet werden kann. Geplant sind mehrere Clients:

### 2.7.1 Native

Der native Client verwendet die Bibliotheken von Windows Forms und soll in C# programmiert werden. Dem Benutzer soll auf einem Blick angezeigt werden, wer gerade im Trainingsraum ist, wie lange dieser schon darin ist und ob er den Raum verlassen darf. Des Weiteren kann der Lehrer in der gleichen Oberfläche einen Besuch eintragen und einen Schüler aus dem Raum entlassen. Benutzern mit administrativen Rechten soll auch eine Verwaltung der Datensätze und eine zeitunabhängige Manipulation er-

möglichst werden. So soll der Administrator einen vollständigen Incident nachträglich eintragen können, ohne dabei an die 15 Minuten Wartezeit zwischen dem Eintragen eines Besuches und dem Entlassen eines Schülers gebunden zu sein. Diese Funktionen werden für „normale“ Benutzer ausgeblendet. Eine vollständige Auflistung der Funktionen befindet sich im Kapitel 2.2 auf Seite 6. Die detaillierten User-Stories befinden sich im Anhang 9.4 auf Seite 77.

### **2.7.2 Mobile**

Die mobilen Clients sollen denselben Funktionsumfang haben wie auch der native Client. So soll es den Lehrern möglich sein flexibel mit einem Tablet die Lokation des Trainingsraumes spontan zu wechseln, ohne dabei an einen fest installierten Rechner gebunden zu sein. Beschäftigt man sich allerdings mit mobilen Anwendungen, wird man feststellen, dass man dabei zumindest für die zwei gängigen Plattformen Android und IOS programmieren muss. Die unterschiedlichen Plattformen führen zu einem erhöhten Wartungsaufwand. Um dies zu verhindern, wird ein Web-Client entwickelt, welcher im nächsten Abschnitt vorgestellt wird.

### **2.7.3 Web**

Der Web-Client soll einen ähnlichen Funktionsumfang wie der nativen Client aufweisen, jedoch diesen als Webseite bereitstellen. Die Daten werden über den vom Backend bereitgestellten RESTful-Service abgerufen und per Asynchronous Javascript and XML (AJAX) angezeigt. Die zu verwendenden Frameworks wurden im Zuge dieser Arbeit für den Einsatz in diesem Umfeld evaluiert und das am besten geeignete ausgewählt. Zur Auswahl standen Active Server Pages (ASP).NET von Microsoft, Java Server Faces (JSF) und Zend. Da der native Client mithilfe des .NET-Frameworks programmiert wird, bietet es sich hier an ASP.NET zu verwenden, da der grundlegende

Code hier wiederverwendet werden kann. Auch ein einheitlicher Programmierstil und eine einheitliche Programmierumgebung auf der Client-Seite spricht für einen Verbleib in der Microsoft-Welt. Zudem bietet es in Verbindung mit Visual Studio eine einfache Einbindung des Design-Frameworks „Bootstrap“. Der Webclient ermöglicht es die Anwendung sowohl auf einem klassischen Desktop-PC zu verwenden, als auch auf mobilen Endgeräten. Durch die Verwendung eines Webclients ist es daher nicht nötig eine Applikation für die unterschiedlichen mobilen Plattformen zu entwickeln, um deren Vorteile für den Anwender zu haben.

## **2.8 VPN**

Dadurch, dass wir die Verwaltung der Daten und ihre Darstellung in ein Backend und ein Frontend unterteilt haben, ist es auch möglich das Frontend von einem anderen Gerät aus aufzurufen. Da die Grundfunktionen der Anwendung nicht durch Passwörter geschützt werden sollen, muss der Datenschutz an einer anderen Stelle realisiert werden. Es soll verhindert werden, dass das Backend aus dem World Wide Web (WWW) erreichbar ist. Dazu darf das Backend grundsätzlich nur auf einer Maschine in einem geschützten Netz, also etwa hinter einer Firewall, installiert werden. Diese Verantwortung liegt allerdings beim Kunden, da dieses Projekt nur aus der Applikation als solches besteht. Um dennoch auch von außerhalb des geschützten Bereiches mit der Anwendung zu kommunizieren dient ein Virtual Private Network (VPN). Mit diesem ist es möglich außerhalb eines geschlossenen Bereiches sich in diesen einzuwählen und als Mitglied des geschützten Bereiches zu agieren. Auch die Anschaffung und Einrichtung eines VPN obliegt dem Kunden selbst, da es für die Anwendung als solches nicht nötig ist.



## 3 Grundlagen

Um das Projekt besser nachvollziehen zu können wird im Folgenden auf die Grundlagen eingegangen, die für das Projekt TRManager relevant sind. Neben den grundlegenden Techniken werden auch die weiterführenden, speziellen Techniken erklärt.

### 3.1 Datenbanken

Unter einer Datenbank versteht man eine Ansammlung von Daten, die logisch als eine Einheit betrachtet werden. Der Unterschied zu einem Dateisystem liegt in einigen Anforderungen an eine Datenbank. Zu diesen Anforderungen gehört etwa das Handling von mehreren konkurrierenden Zugriffen. Wird allgemein von einer Datenbank gesprochen, ist meist ein Datenbankmanagement-System (DBMS) gemeint. Ein DBMS enthält kann mehr als eine Datenbank verwalten. Die Datenbank selbst steht in diesem Kontext lediglich für die gespeicherten Informationen und deren Meta-Daten. Wird also von der Datenbank gesprochen, muss dieser Begriff im Kontext betrachtet werden. Neben dem Handling von konkurrierenden Zugriffen stellt ein DBMS weitere Features zur Verfügung. Dazu gehört standardmäßig auch die Implementierung von Regeln zur Zugriffssteuerung. So lässt sich die Datenbank gegen unerlaubte Manipulation oder lesende Zugriffe schützen. Ein DBMS liefert oft auch Backup- und Recovery-Verfahren mit. Um ein DBMS zu nutzen wird eine Benutzerschnittelle geboten. So kann man das DBMS zum einem direkt über eine grafische Benutzeroberfläche steuern, aber auch über Schnittstellen wie etwa Open Database Connectivity (ODBC). ODBC verwendet hierzu SQL (Eigenständiger Begriff, kann aber als Abkürzung von Structured Query Language gesehen werden) zur Kommunikation mit dem DBMS. Eine weitere grundlegende Anforderung an ein DBMS ist die Vermeidung von Redundanz, sowie die Sicherstellung einer maximalen Integrität. Sind Informationen mehr als einmal in einer

Datenbank gespeichert, ist es schwer sicherzustellen, dass diese alle den gleichen Stand haben, wenn die Information geupdatet wird. Diesem Umstand kann bereits durch eine gute Planung der Datenbank entgegen gewirkt werden. Die Datenintegrität bezeichnet die Korrektheit der Daten in Bezug auf die Realität. Ein Beispiel hierfür ist die Angabe eines Tages. „abc“ als Wert wäre ein zu vermeidender Eintrag. Dies lässt sich erzwingen, in dem beispielsweise ein Datumsfeld auch lediglich Werte in einem Datumsformat akzeptiert.

## 3.2 REST

REST bezeichnet ein Paradigma für verteilte Systeme. Grundgedanke von REST ist es, einen Service über eine bereits genormte und universelle Schnittstelle (HTTP) anzubieten. Der Vorteil von REST liegt darin, dass man, wie oben genannt, seine Services über eine uniforme Schnittstelle anbieten kann, die sehr weit verbreitet ist. Dies erleichtert das Konsumieren des Services, da man bereits bekannte Methoden(im Sinne von Methodik) verwenden kann, um auf diese zuzugreifen, und auch die Struktur der HTTP-Anfragen gut bekannt ist. Zudem ist die Infrastruktur, die für REST nötig ist, im WWW sehr weit ausgebaut (HTTP-fähige Clients, Web- und Anwendungsserver, HTML-Parser) und viele der schon bestehenden Dienste sind bereits RESTful (Auslieferung der Seiten über HTTP, Verwendung von HTTP-Methoden wie GET und POST etc.)

REST ist zustandslos, das bedeutet dass jede Nachricht alle Informationen bereithält um diese zu verstehen. Dies begünstigt die Skalierbarkeit solcher Services, da man eingehende Anfragen ohne große Probleme auf andere Systeme umleiten kann (zur Lastverteilung), da der Server keine Zustandsinformationen zu einer Verbindung speichert. Diese werden ausschliesslich clientseitig vorgehalten, der Service dient hier sozusagen als Datenquelle. Neben der guten Skalierbarkeit bietet REST die Vorteile von HTTP

wie zum Beispiel die einfache und erprobte Verschlüsselung über Transport Layer Security (TLS) sowie HTTP-Caching.

Daten werden bei REST hauptsächlich über Formate wie JSON und XML repräsentiert, dazu kann der REST-Service je nach Client und angeforderter Formatierung ein anderes Format wählen.

### 3.3 Java

Java ist eine vom Unternehmen Sun Microsystems entwickelte, objektorientierte Sprache. 2010 wurde Sun Microsystems von Oracle aufgekauft und wird seitdem von Oracle vertrieben. Die Programmiersprache ist Kern der Java-Technologie, neben der Sprache gibt es die Entwicklungswerkzeuge (JDK, Java Development Kit) und die plattformunabhängige Laufzeitumgebung (JRE, Java Runtime Environment). In der Laufzeitumgebung sind die JVM (Java Virtual Machine) und die mitgelieferten Bibliotheken enthalten.

Das Ziel der JVM, also im Grunde die Virtualisierung der Laufzeitumgebung, ist die plattformunabhängigkeit. So können Java-Programme auf jedem System mit einer JRE beziehungsweise JVM ausgeführt werden. Von Oracle selbst werden JREs für Linux, OS X, Solaris und Windows angeboten, aber andere Hersteller lassen sich ihre eigenen Laufzeitumgebungen für die jeweilige Plattform zertifizieren. So gibt es zum Beispiel Laufzeitumgebungen für Hifi-Anlagen, Autos und viele mehr.

Java ist aktuell in Version 8.92 (8 Update 92) verfügbar. <sup>2</sup>.

---

<sup>2</sup> Bernhard Steppan. *Einstieg in Java 7: [eine umfassende und professionelle Einführung; mit vielen Beispielen und kommentierten Lösungen; Programmierung von GUIs, Datenbanken, dynamischen Websites u.v.m.]*. Galileo Press, Bonn, 4., aktualisierte Aufl. edition, 2012, Kapitel 4

### 3.3.1 JUnit

JUnit ist ein Framework zum Erstellen von automatisierten Tests. Es bietet die Möglichkeit, automatisierte Tests an sogenannten Units (Klassen beziehungsweise Methoden) zu implementieren und durchzuführen. Es wurde ursprünglich von Erich Gamma und Kent Beck entwickelt und ist an das Test-Framework SUnit (Smalltalk) angelehnt. Dabei ist JUnit nicht nur für Java verfügbar, jedoch werden diese nach der Programmiersprache benannt, für die sie verwendet werden und werden unter dem Begriff „xUnit“ zusammengefasst.

Zur Evaluierung eines Tests gibt es in JUnit nur zwei Zustände: Entweder er gelingt oder er schlägt fehl (Failure beziehungsweise Error), was mit einer Exception signalisiert wird. Der Unterschied zwischen Failure und Error ist der, dass Failure erwartet werden kann, wobei Errors eher unerwartet auftreten.

Zudem existieren für JUnit zahlreiche Visualisierungs-Tools, die auch Dinge wie etwa Testabdeckung überprüfen. Zu diesen Tools zählt zum Beispiel Jacoco.

## 3.4 Spring

Das Spring-Framework ist ein quelloffenes Framework für die Java-Plattform. Es soll die Entwicklung mit Java/JavaEE vereinfachen und gute Programmierpraktiken fördern. Es wird zur Entwicklung von modernen, Java-basierten Enterprise-Anwendungen genutzt und bietet ein leicht verständliches Programmier- und Konfigurationsmodell. Dabei sind die meisten Komponenten voneinander entkoppelt, dies führt zu einer guten Trennung von zum Beispiel Geschäftslogik und der darunter verwendeten Umgebung.

Spring bietet Dependency Injection, Aspektorientierte Programmierung inklusive deklarativem Transaktionsmanagement und Unterstützung von Java Persistence API (JPA), Java Database Connectivity (JDBC), Java Message Service (JMS) und anderen.

### 3.4.1 Spring Boot

Spring Boot bietet eine vollständige, lauffähige Umgebung für einfache Anwendungen, die ohne Konfiguration auskommen und alle Bibliotheken enthält. Es eignet sich gut für kleinere, überschaubare Anwendungen, in denen keine komplexen Konfigurationsmöglichkeiten benötigt werden.

### 3.4.2 Spring MVC

Spring MVC bietet dem Anwendungsentwickler alle benötigten Funktionen, um einen Web-Service anzubieten. Darunter fallen zum Beispiel die Controller (Controller), Request-Mappings (Zuordnen von Requests zu Controllern, RequestMapping) und anderen. Diese behandeln Anfragen und stellen die Schnittstelle zur Aussenwelt dar.

### 3.4.3 Spring Data JPA

Spring Data JPA beinhaltet die Klassen für das Objektrelationale Mapping sowie für Repositories. Es ist die Zwischenschicht zwischen der Anwendung und der Datenbank und übernimmt die Zuordnung der Daten zu den Objekten. Im Fall dieser Arbeit wird Hibernate als JPA-Implementierung verwendet.

**3.4.3.1 Hibernate** Hibernate ist ein Persistenz- und ORM-Framework für Java. Es wird verwendet, um objektrelationale Abbildungen in relationalen Datenbanksystemen umzusetzen.

Hierzu bildet es eine Abstraktionsschicht zwischen der Datenbank und dem Anwendungsentwickler. So muss sich der Benutzer zum Größtenteil nicht mehr mit dem Mapping und der Datenbank an sich beschäftigen und kann gegen die Schnittstelle von Hibernate programmieren. Diese generiert dann die benötigten SQL-Anweisungen für

das darunterliegende Datenbanksystem.

Hibernate ist kein Teil von Spring, wird aber in dieser Arbeit in Spring eingebunden<sup>3</sup>.

#### 3.4.4 Dependency Injection

Dependency Injection beschreibt in der objektorientierten Programmierung ein Entwurfsmuster, das die Abhängigkeiten eines Objekts erst zur Laufzeit festlegt. Werden bei der Erzeugung eines Objektes andere Objekte benötigt, sind diese Abhängigkeiten zentral gespeichert und werden nicht selbst vom erzeugten Objekt erzeugt.

Spring setzt diese mit dem sogenannten IoC-Container um. Inversion of Control (IoC) beschreibt die „Umkehr der Steuerung“. Hierbei wird die Steuerung der Anwendung an das Framework abgegeben, die Anwendung selbst wird nur durch dieses aufgerufen. Implementiert wird dies über die „@Autowired“-Annotation, dies teilt Spring mit dass hier die Injektion gewünscht ist. Spring verwaltet dann die Zuweisung dieser Objekte<sup>4</sup>.

#### 3.4.5 Aspect Oriented Programming

Aspektorientierte Programmierung(AOP) beschreibt ein Programmierparadigma in der objektorientierten Programmierung. Das Ziel der AOP ist es, logische Aspekte (oder generische Funktionalität) von der eigentlichen Geschäftslogik zu trennen. Entwickelt wurde dieses Konzept bei Xerox PARC von Gregor Kiczales und seinem Team. 2001 wurde dort auch die erste Sprache dafür vorgestellt, AspectJ, eine aspekt-orientierte Erweiterung von Java. Im Rahmen von Spring wird AOP für Funktionen wie Logging, Deklaratives Transaktionsmanagement und andere verwendet<sup>5</sup>.

---

<sup>3</sup> Dave Minter et al. *Beginning Hibernate*. Apress, New York, 3rd ed (online-ausg.) edition, 2006

<sup>4</sup> Jan Machacek et al. *Pro Spring 2.5*. Apress, Berkeley, Calif., 2008, Kapitel 3

<sup>5</sup> Jan Machacek et al. *Pro Spring 2.5*. Apress, Berkeley, Calif., 2008, Kapitel 5

### 3.5 .NET Framework

Das .NET-Framework ist Teil der Software-Plattform .NET und wurde von Microsoft entwickelt. Ähnlich der Java-Umgebung bietet das .NET-Framework eine Laufzeitumgebung, die Common Language Runtime, in der die Programme ausgeführt werden. Neben der Laufzeitumgebung wird eine Sammlung von Klassenbibliotheken, API's und Dienstprogrammen.

.NET unterstützt eine Vielzahl von Programmiersprachen, die vom Compiler in eine Zwischensprache übersetzt werden, die sogenannte Common Intermediate Language und bei der Ausführung dann in die eigentliche Maschinensprache des Zielsystems übersetzt. Dies ermöglicht Sprachneutralität, also das .NET-Framework mit unterschiedlichen, an die CLR angepassten Sprachen zu programmieren.

Aufgrund der Verwendung einer virtuellen Maschine ist Plattformunabhängigkeit grundsätzlich gegeben. Es gab auch Versuche diese auf anderen Plattformen anzubieten, die Unterstützung dieses Projekts (Shared Source CLI, SSCLI) für Mac OS und FreeBSD wurde aber nach Version 2.0 eingestellt. Im Bereich Open-Source existieren mehrere Projekte mit diesem Ziel, mit dem bekanntesten Vertreter Mono.

#### 3.5.1 Windows Forms

Windows Forms ist eine Application Programming Interface (API) zur Erstellung von graphischen Benutzeroberflächen und ist Teil des .NET-Frameworks. Es bietet dem Anwendungsentwickler Zugriff auf Elemente zur Erstellung von Microsoft-Windows-Benutzeroberflächen. Um dies zu erreichen, wird die existierende Windows-API gewrappt und so zur Verfügung gestellt.

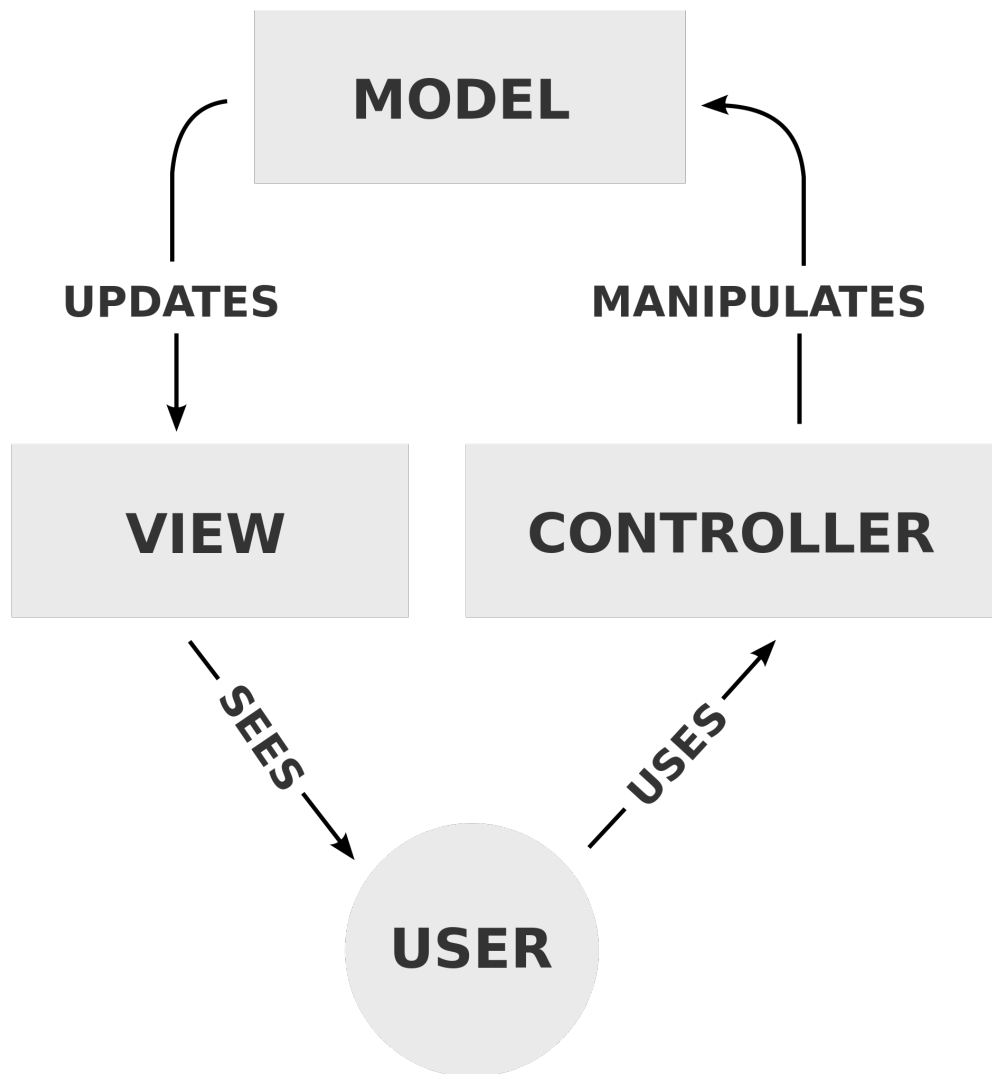
### 3.5.2 ASP.NET

ASP.NET ist eine von Microsoft entwickelte Technologie, welche auf dem .NET Framework basiert. Dieses Framework besteht aus einer Vielzahl von Klassenbibliotheken und Schnittstellen. Von den vielen unterstützten Programmiersprachen sind C# und VB.NET die meist verwendeten. Aufgrund des Projektes wird im Rahmen dieser Arbeit nur ASP.NET in Zusammenhang mit C# betrachtet. ASP.NET löste 2002 den Vorgänger ASP ab. Der Marktanteil von Webseiten, die mit ASP.NET realisiert wurden, liegt derzeit bei ca. 16% (2) (Stand 2016). Damit belegt ASP.NET den zweiten Platz der meistverwendeten Websprachen nach PHP, die ca. 81% der Webseiten verwenden. Das .NET Framework ist die Schnittstelle zwischen dem Programmcode und dem Betriebssystem, auf dem das Projekt läuft. Viele Funktionen, welche man ansonsten selbst schreiben musste, sind im Framework bereits enthalten und lassen sich nutzen. Durch diesen Umstand wird beim Kompilieren der Programmcode in eine Zwischensprache umgewandelt, welche sich Microsoft Intermediate Language (MSIL) nennt. So lassen sich auch in einem Projekt unterschiedliche Programmiersprachen verwenden.

### 3.5.3 Model View Controller (MVC)

Der Begriff MVC steht für ein Architekturmuster zum strukturierten und dennoch flexiblen Programmentwurf. So soll eine spätere Änderung oder Erweiterung erleichtert und eine Wiederverwendbarkeit der einzelnen Komponenten ermöglicht werden. Die einzelnen Programmaspekte werden dabei den folgenden drei Komponenten zugeordnet. Abbildung 1 auf Seite 21 zeigt die Interaktionen in einer MVC-Anwendung.



Abbildung 1: Interaktionen im MVC-Modell<sup>a</sup>

<sup>a</sup> <https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png>

### 3.5.3.1 Model (Modell)

Zu den Modellen gehören die Klassen, welche die Daten repräsentieren. In diesen Klassen ist auch die Validierungslogik implementiert welche zur Einhaltung der Geschäftsregeln dient. Dieser Abschnitt ist von der Präsentation und der Steuerung unabhängig.

### 3.5.3.2 View (Präsentation)

Im Allgemeinen stellt eine View Teile der Daten aus den Modellen in einer Benutzeroberfläche zu Verfügung. Bei ASP.Net sind dies die Vorlagen - Dateien welche von der Anwendung benutzt werden um dynamisch HTML-Antworten zu generieren.

### 3.5.3.3 Controller (Steuerung)

Die Controller Klassen analysieren die eingehenden Anfragen und leiten Diese an das entsprechende Objekt der Model-Schicht. Ein Controller agiert demnach als Vermittler zwischen den Schichten und hat außerdem die Möglichkeit den Funktionsumfang des Nutzers auf der Präsentationsebene einzuschränken.

## 3.6 Logischer Aufbau einer Anwendung

Konventionen helfen Menschen sich zu orientieren und so haben sich auch einige Konventionen im Anwendungs-Design entwickelt, die eingehalten werden sollten, um die Nutzbarkeit einer Anwendung zu erhöhen. Dazu gehört zum Beispiel, dass ein abspielbares Video das bekannte Dreieck des Abspiel-Knopfes enthält, damit der Nutzer dieses anklicken kann. Es muss für den Nutzer erkennbar sein, welche Elemente klickbar sind. Dies muss nicht wie auf alten Webseiten mit blauer Schriftfarbe und unterstrichenem Text passieren, aber für den Nutzer muss es erkennbar sein. Weiterhin muss Inhalt entfernt werden, der die Person davon ablenkt ihre Aufgabe zu erledigen. So fällt es beispielsweise schwer Angebote auf einer Produktwebseite zu erkennen, wenn alle Produkte mit großen Texten versehen sind, die sagen „Klick hier für dein Sonderangebot“, da der Nutzer dann nicht mehr weiß wo er anfangen soll und vielleicht die Seite sogar wieder verlässt. Und zum Schluss hilft eine gute Navigation. So sollte im Kopfbereich der Seite oder seitlich immer die Hauptnavigation klar erkenntlich sein und immer gleich beziehungsweise sehr ähnlich aussehen, um dem Nutzer auch die Sicherheit zu geben, dass er sich noch immer auf der gleichen Seite befindet.

### 3.7 Web-Applikationen

Als Grundlage für dieses Kapitel ist die Definition von Matthias Rohr aus seinem Buch „Sicherheit von Webanwendungen in der Praxis“ gewählt. Diese besagt: „Eine Webanwendung ist eine Client-Server-Anwendung, die auf Webtechnologien (HTTP, HTML etc.) aufsetzt.“<sup>6</sup> . Eine Webanwendung wird in der Regel über einen Webbrowser aufgerufen, in dem die serverseitige Anwendung dargestellt wird. Es ist aber auch möglich auf andere Weise auf eine Webanwendung zuzugreifen, etwa über eine Kommandozeile. Daher wird in der Regel vom Client oder aber User Agent gesprochen.

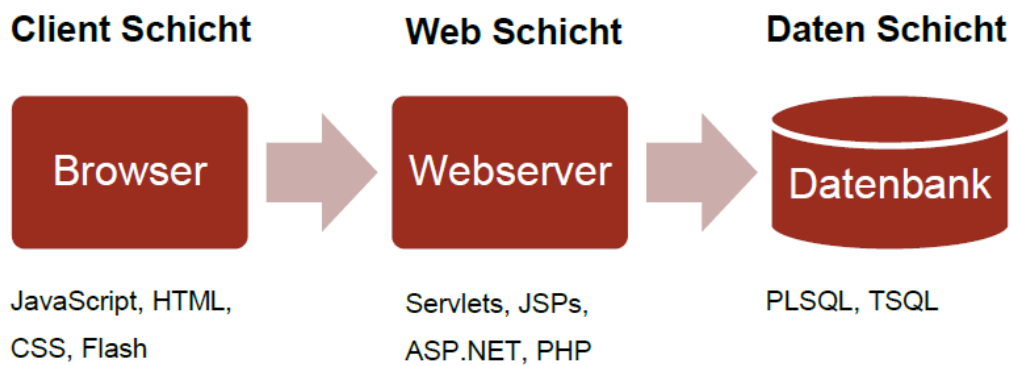


Abbildung 2: Schichtenarchitektur einer Webanwendung (eigene Darstellung)

Serverseitig wird die Anwendung auf einem Webserver oder Applikationsserver ausgeführt. Um dies nicht unnötig kompliziert zu gestalten wird im Rahmen dieser Arbeit verallgemeinert von einem Webserver gesprochen. Dieser kann in der Praxis unter anderem auch für ganze Cluster aus Webservern und Applikationsservern bestehen. Dieser Webserver greift nun seinerseits in der Regel auf Hintergrundsysteme wie etwa Datenbanken zu. So entsteht die so genannte 3-Schichtenarchitektur, welche in Abbildung 2 auf Seite dargestellt ist. Zur Veranschaulichung wurden exemplarisch Technologien den einzelnen Schichten zugeordnet. Mittlerweile reicht diese Darstellung kaum mehr

<sup>6</sup> Matthias Rohr. *Sicherheit von Webanwendungen in der Praxis*. Springer Vieweg, 2015

aus, da deutlich mehr Schichten benötigt werden, etwa weil auf deutlich mehr als ein Hintergrundsystem zugegriffen wird und auch der serverseitige Teil der Anwendung separat gegliedert ist. Für dieses doch recht überschaubare Projekt des TRManagers reicht das 3-Schichten-Modell allerdings zum Verständnis.

### **3.8 Frameworks**

Ein Framework ist eine Art Rahmen, der Entwicklern ein Grundgerüst für bestimmte Funktionen bietet. Dabei ist ein Framework selbst kein eigenständiges Programm, es bietet Funktionen und gibt eine bestimmte Struktur vor, die eigentliche Anwendung wird unter Verwendung dieses Gerüsts implementiert. Frameworks unterstützen den Programmierer auch bei der Umsetzung des Don't repeat yourself (DRY)-Prinzips, so dass Wiederholungen vermieden werden. Folgend werden die Frameworks Bootstrap für die Seite des Clients und für die Serverseite Alternativen zu ASP.NET dargestellt.

#### **3.8.1 Bootstrap**

Bei Bootstrap handelt es sich um ein freies CSS/HTML/JavaScript-Framework. Bootstrap wurde ursprünglich von Twitter als internes Tool zu Vereinheitlichung der Oberflächenprogrammierung entwickelt. In Rahmen dieses Projektes merke Twitter, dass sich mit diesem Tool deutlich mehr gestalten lässt. So sorgt dieses Tool nicht nur für ein einheitliches Aussehen, sondern auch für eine Reduzierung der Entwicklungszeit, da die Toolkits das Design erheblich vereinfachten. Die Vorlagen in den neueren Versionen von Visual Studio enthalten bereits von vornherein Bootstrap. Die wesentlichen Eigenschaften des Designs wie etwa die Text- und Hintergrundfarben oder aber auch die einzelnen Größen sind in der „config.json“ - Datei festgelegt. Diese Datei lässt sich auf der Webseite von Bootstrap eigenhändig zusammenstellen.

### **3.8.2 Zend**

Das Zend-Framework ist ein unter der BSD-Lizenz veröffentlichtes Framework für PHP, veröffentlicht vom Unternehmen Zend Technologies. Es ist komponenten-orientiert, viele Teile sind voneinander unabhängig. So kann Zend auch in Kombination mit anderen Lösungen verwendet werden. Es ist komplett objektorientiert umgesetzt und bietet dem Programmierer zum Beispiel Funktionen wie Dependency Injection. Komponenten aus dem Zend-Framework enthalten unter anderem ein MVC-System, Authentifizierung und Benutzerverwaltung sowie verschiedene Hilfskomponenten wie zum Beispiel die Erstellung von PDF-Dateien und Email-Versand.

### **3.8.3 Java Server Faces**

JSF ist ein Framework für die Entwicklung von grafischen Benutzeroberflächen im Bereich von Webapplikationen und somit eine Alternative zu ASP.NET. Die zugrundeliegenden Techniken sind hier Servlets und JSP. Damit gehört JSF zu den Webtechnologien von Java Enterprise Edition. JSF erlaubt es dem Anwendungsentwickler, Komponenten für Benutzerschnittstellen in Webseiten einzubinden. Zudem trennt JSF strikt zwischen Modell, Präsentation und Steuerung (Model View Controller) um Anwendungen besser zu strukturieren.

Zur Entwicklung von JSF-Anwendungen wird ein JDK sowie ein Servler-Container wie zum Beispiel Apache Tomcat vorausgesetzt.

## 4 Umsetzung des Backends

Das Backend soll wie in Kapitel 2.3 beschrieben die Datenquelle für die verschiedenen Clients sein. Umgesetzt werden soll dies über einen RESTful Service. Es wird das Framework „Spring“ eingesetzt. Dies erlaubt eine einfache Erstellung von Web-Services und unterstützt den Programmierer mit Werkzeugen wie Hibernate(Hibernate ist ein externes Tool und gehört nicht direkt zu Spring), die den Zugriff auf der Datenbank stark vereinfacht und Änderungen an der Modellierung direkt in der Datenbank umsetzt.

### 4.1 Anforderungen

Das Backend soll den Clients die Daten wie oben genannt über einen RESTful-Service zur Verfügung stellen. Dies vereinfacht den Zugriff auf die Ressourcen und ist leicht in bestehende Infrastrukturen einzubinden. Das Backend soll nach Möglichkeit über Hyper Text Transfer Protocol Secure (HTTPS) zur Verfügung stellen, damit übertragene Daten auf dem Weg gesichert sind. Eine Authentifizierung auf Zugriffsebene ist optional, da der Service in einem geschlossenen Netz betrieben werden soll. Das Backend soll zudem Authentifizierungsanfragen auf Anwendungsebene bearbeiten können.

### 4.2 Implementierung

Das Backend wird mit dem quelloffenen Framework Spring umgesetzt. Spring bietet den Anwendungsentwickler Funktionen wie Dependency Injection und ähnliche (vergleiche Kapitel 3.4 auf Seite 16), die die Implementierung eines Webservices erheblich vereinfacht. Die für dieses Projekt verwendeten Module umfassen Spring Boot, Spring Data und Spring MVC. Spring Boot erlaubt die einfache Erstellung sofort lauffähiger

Spring-Anwendungen, da hier keine komplexen XML-Konfigurationen vorgenommen werden müssen, und das Projekt übersichtlich genug, um keine komplexen Konfigurationen zu benötigen.

#### 4.2.1 Model

Das Model legt die Datenstrukturen fest, diese stellen die Objekte in der Anwendung dar wie zum Beispiel eine Klasse oder einen Schüler. Ebenso legen diese über die annotierten Model-Klassen die Datenbankstruktur fest (Siehe Kapitel „Hibernate“). Für die Darstellung der Daten werden folgende Klassen benötigt:

**LazyObject** Die LazyObject-Klasse ist die Basis-Klasse aller anderen Model-Klassen.

Von dieser erben alle anderen Klassen. Sie legt die Grundstruktur jedes Objektes fest, wie zum Beispiel die ID. (Siehe Anhang, Seite 68)

**Teacher** Die Teacher-Klasse stellt jeweils einen Lehrer dar. Jeder Lehrer hat einen Namen und eine Abkürzung.

**Form** Die Form-Klasse stellt jeweils eine Schulklasse dar. Jede Klasse hat einen Klassenlehrer, einen Namen und eine Liste mit allen in dieser Klasse befindlichen Schülern.

**Student** Die Student-Klasse stellt jeweils einen Schüler dar. Gespeichert werden der Name des Schülers, geteilt in Vor- und Nachname, die Klasse in der der Schüler ist und eine Liste aller Vorfälle, in die der Schüler verwickelt ist.

**Incident** Die Incident-Klasse stellt jeweils einen Vorfall dar. Jeder Vorfall speichert den schickenden und den beaufsichtigenden Lehrer, den zugehörigen Schüler, eine interne Vorfallsnummer, die Ankunftszeit und Entlassungszeit sowie einen Kommentar, der den Vorfall selbst beschreibt und frei eingegeben werden kann.

**Comment** Die Comment-Klasse stellt jeweils einen vorgefertigten Kommentar dar, der von den Clients verwendet werden kann um das Eintragen eines neuen Vorfalls zu vereinfachen.

## Datenbank

Die Tabellen werden von Hibernate (Siehe Kapitel 3.4.3.1, Seite 17) aus dem Objektmodell (Beschrieben in Kapitel 3.5.3.1 auf Seite 21) generiert. Hierzu werden die Tabellennamen, Beziehungen und die Optionen wie zum Beispiel „Not Null“ mithilfe von Annotationen an Hibernate weitergegeben. Aus diesen Informationen erstellt Hibernate das relationale Datenmodell (Siehe Abbildung 3).

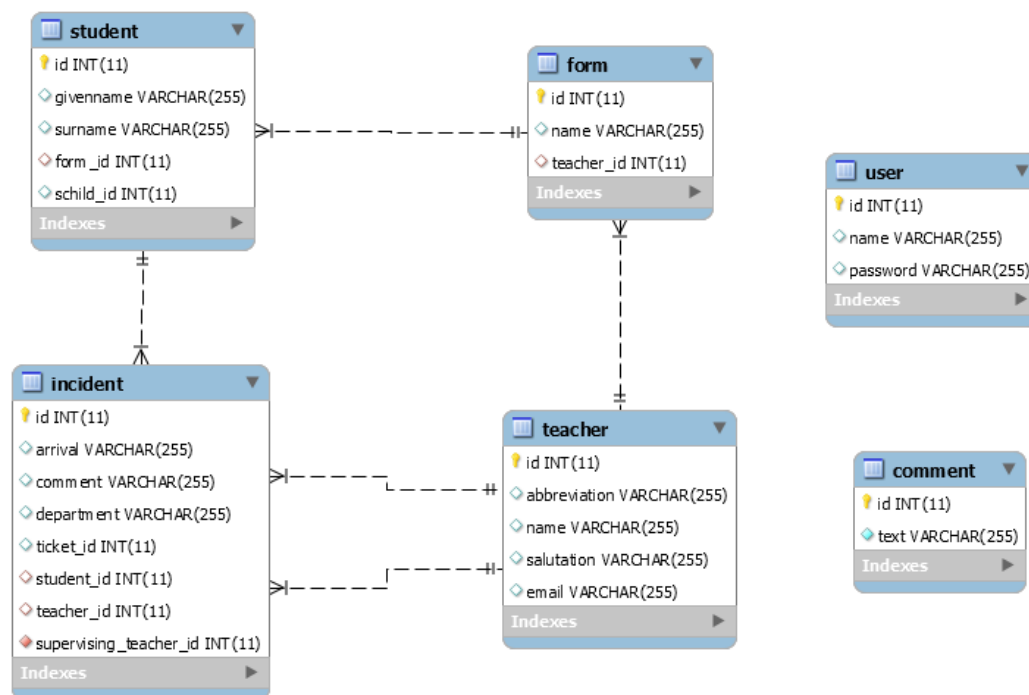


Abbildung 3: ER-Modell der Datenbank

**teacher** Die „teacher“-Tabelle hält Daten über die Lehrer vor. Dies umfasst den vollen Namen (name), die Abkürzung (abbreviation) und die Anrede (salutation)



**form** Die „form“-Tabelle hält Daten über die Schulklassen vor, dies beinhaltet die Bezeichnung der Klasse (name) und einen Fremdschlüssel „teacher\_id“, der den Klassenlehrer referenziert.

**student** Die „student“-Tabelle hält Daten über die Schüler, die im System erfasst sind, vor. Dies umfasst den Namen des Schülers geteilt in Vor- und Nachname (givenname, surname) und einen Fremdschlüssel „form\_id“, die die Schulkasse, der der Schüler angehört referenziert. Zudem wird die Identifikationsnummer des Schülers gespeichert, um gleichnamige Schüler eindeutig identifizieren zu können (schild\_id).

**incident** In der „incident“-Tabelle werden alle Vorfälle gespeichert, die über den TRManager eingetragen werden. Dazu gehören die Ankunftszeit (arrival), der Zeitpunkt der „Entlassung“ (department), den zugehörigen Schüler (Foreign Key (FK) student\_id), den schickenden Lehrer (FK teacher\_id), ein Kommentarfeld (comment) und eine interne Vorfallsnummer, die Ticket ID(ticket\_id).

**comment** In der „comment“-Tabelle werden vorgefertigte Antworten für Vorfälle gespeichert. Das Feld „text“ hält hierbei jeweils einen vorgefertigten Kommentar vor, der von den Clients abgerufen werden kann.

#### 4.2.2 Repositories (Spring Data)

Ein Repository stellt im Grunde die Verbindung zwischen der Datenbank (Hibernate) und der Anwendung selbst da. Es ist für die Zusammensetzung der Queries zuständig und liefert Daten zurück und schreibt Daten in die Datenbank. Werden hier Änderungen an einem Objekt vorgenommen, werden diese

transparent in der Datenbank geändert, hier ist also Vorsicht geboten. Um fehlerhafte Änderungen durch unsachgemäßen Gebrauch der Repositories zu vermeiden, wird normalerweise eine Schicht dazwischen geschoben, um davon zu abstrahieren(Siehe

Kapitel 4.2.3). Standardmäßig werden die Repositories von Spring mit vordefinierten Zugriffsmethoden generiert, um die Handhabung zu vereinfachen. Es ist aber auch möglich, andere Queries als die von Spring generierten zu verwenden, zum Beispiel falls die per Standard gegebenen Zugriffsmöglichkeiten von Spring Data nicht ausreichen. Dies ist jedoch im Rahmen des TR-Managers nicht notwendig, da der Datenbestand relativ klein und auch keine komplexen Abhängigkeiten im Datenmodell bestehen.

```
1 public interface FormRepository extends JpaRepository<Form,Integer>{  
    @Query("SELECT r from Form r where r.ID = ?1")  
3    Form findFormById(int id);  
}
```

Listing 1: Definition eines Repositories

In Listing 1 sieht man die Definition des Repositories für die Form-Objekte. Das Repository ist im Grunde ein Interface, das von „JpaRepository“ erbt und mit Generiken versehen wird, um den Rückgabetyt und den „Suchwert“ (In dem Fall Integer, die ID) festzulegen. Im Beispiel ist auch eine selbst-definierte Query zu sehen, diese werden mit der „Query“-Annotation festgelegt und werden von Spring vorrangig zur Abfrage verwendet. Für die restlichen Methoden (findAll, findById, delete etc.) werden die von Spring Data/Hibernate generierten Queries verwendet.

### 4.2.3 Services

Die Services sind wie in Kapitel 4.2.2 beschrieben die Abstraktion der Repositories für die restliche Anwendung und behandeln Ausnahmefälle, damit diese nicht an die Repositories weitergereicht werden. Im einfachsten Fall reicht der Service die im Repository gefundenen Daten weiter an die höheren Komponenten, damit nicht direkt auf das Repository zugegriffen wird und direkte Manipulationen der Datenbank mi-

nimiert werden. Dies sollte ausschließlich über die „save“-Methode der Repositories durchgeführt werden. Weitere Funktionen die die Services erfüllen könnten wären höhere Konsistenzprüfungen (Abhängigkeiten, die nicht auf Datenbankebene definiert werden können beziehungsweise in der Geschäftslogik umgesetzt werden sollen) sowie die Validierung von Daten. Dies wird in der Anwendung jedoch nicht benötigt, deshalb wurde darauf verzichtet.

```
@Service
2 public class FormServiceImpl implements FormService{
    @Autowired
4     private FormRepository repository;

6     @Override
    public Form getFormById(int id){
8         Form form = repository.findOne(id);

10         if(form == null) return null;
        return repository.findFormById(id);
12     }
}
```

Listing 2: Definition eines Services

In Listing 2 wird eine beispielhafte Definition eines Services mit einer Methode gezeigt. Die „Service“-Annotation markiert diese Klasse für Spring als Service. Das Repository-Objekt dient als Datenquelle für die Anfragen, diese wird durch den IoC-Container injiziert und steht automatisch zur Laufzeit zur Verfügung (So wie alle Objekte mit „Autowired“-Annotation). Die beispielhafte Methode „getFormById“ sucht eine Schulklasse anhand ihrer ID und liefert diese zurück.

#### 4.2.4 Controller

Die Controller sind die Anlaufstelle für Anfragen über das Netzwerk und bearbeiten diese. Sie bedienen die Clients und liefern ihnen die angeforderten Daten zurück. Die in dieser Arbeit verwendeten Controller werden von Java (beziehungsweise Spring MVC) (@RestController) bereitgestellt und erlauben eine einfache Zuordnung zu einer Uniform Resource Identifier (URI), unter der sie ansprechbar sind sowie die Zuordnung der Methoden, die die unterschiedlichen Anfragen (zum Beispiel GET, POST, PUT oder DELETE) bearbeiten.

#### Aufbau der Controller

Im Allgemeinen gibt es für jede Model-Klasse einen Controller, der Daten zum jeweiligen Model liefern kann. Es können jedoch noch weitere Controller(oder weniger) definiert werden, die mehr oder weniger Funktionalität sowie zusätzliche, vom Datenmodell abgekoppelte, Funktionen anbieten können. So wurde zum Beispiel zur allgemeinen Verwaltung des Datenbestandes ein Backup-Controller implementiert, der das clientseitige Backup unterstützt (alle Daten ausliefern, Daten löschen und die Tabellen in den Anfangszustand versetzen)

```
1  @RestController
   @RequestMapping(value = "/trmanager/form")
3  public class FormController {
       @Autowired
5      private FormService service;

7      @RequestMapping(method = RequestMethod.GET)
      public List<Form> getAllForms(){
9          List<Form> forms = service.getAllForms();
          return forms;
11     }
}
```

Listing 3: Definition eines Controllers

In Listing 3 sieht man die einfache Definition eines Controllers. Die Klasse wird mit der „@RestController“-Annotation versehen, die dem IoC-Container mitteilt, dass diese Klasse als REST-Controller definiert ist. Die Annotation „@RequestMapping“ legt die Adresse fest, auf der der Controller lauscht (In diesem Fall „<hostname>:port/trmanager/form“) und Anfragen entgegennimmt. Die in der Klasse definierte Beispielmethode liefert auf eine GET-Anfrage alle Form-Objekte (definiert über „method=RequestMethod.GET“) aus.

#### 4.2.4.1 Funktionsumfang der Controller

Jeder Controller der einer Model-Klasse (Form, Teacher, Student, Incident, User, Comment) zugeordnet ist muss mindestens die folgenden Methoden umsetzen:

- Ausliefern der Daten (getById & getAll)
- Hinzufügen von einzelnen Datensätzen (add)

- Hinzufügen von mehreren Datensätzen (addBulk)
- Manipulation der Daten (edit)
- Löschen von Datensätzen (delete)

Diese werden benötigt, um den Clients die nötigen Funktionen zu bieten, um die Anforderungen an das Programm umzusetzen.

### 4.3 Tests

Die Tests werden dazu verwendet um die Integrität und die Validität des Backends zu überprüfen. Hierzu wird JUnit verwendet, (Siehe Kapitel 3.3.1 auf Seite 16) um automatisierte Unit-Tests durchzuführen und die Funktionalität des Backends zu gewährleisten.

Zu Beginn der Tests werden Testdatensätze in die Datenbank geladen (Von der produktiven Datenbank getrenntes Schema) und auf diesen werden dann die Tests durchgeführt. Getestet werden die Controller (Anfragen über das Netzwerk) und die Services beziehungsweise die Repositories. Bei den Controllern wird die Richtigkeit der HTTP-Antworten geprüft sowie die Authentifizierung. Die Services und Repositories werden insoweit geprüft, dass sie die Datensätze anhand der implementierten Logik zurückgeben sowie richtig in die Datenbank übernehmen.

#### 4.3.1 Controller-Tests

Da die Controller die Schnittstelle zur Aussenwelt darstellen, ist hier die Richtigkeit der Antworten besonders wichtig. Um die einzelnen Methoden zu validieren wird jeder Controller und jede Methode eines Controllers überprüft. Zu den Überprüfungen gehören das Handling von Anfragen mit „falschen“ Daten, die Richtigkeit der zurückgelieferten Daten und falsche Anfragen. Da die jeweiligen Controller sehr ähnlich aufgebaut sind,

lässt sich hier vieles übernehmen was auch für einen anderen Controller benutzt wurde. Zu Beginn eines jeden Tests wird die Datenbank neu geladen, das heisst die Testdaten, wie oben beschrieben, werden wiederhergestellt, damit sichergestellt ist dass bei Inserts keine Fehler bezüglich Duplikaten auftreten, die referenzierten Datensätze vorhanden sind etc.

Die Tests werden mit MockMVC aus dem Spring Integration Testing durchgeführt. Basis für die Test bilden die Web-Requests, die sich aus dem MockMVC-Objekt generieren lassen (RequestBuilder). Diese werden dazu verwendet, um Anfragen an das Backend abzusetzen und so zu überprüfen. So kann zum Beispiel getestet werden, ob der Server richtig reagiert, wenn ein Objekt nicht gefunden werden kann (In dem Fall 404: Not found). Falls das erwartete Verhalten eintritt, ist der Test erfolgreich. In dem Stil werden alle Tests für jede Methode der Controller durchgeführt, es werden alle möglichen Ergebnisse der Controller „erzwungen“. Im Fall dieser Anwendung gibt es für jeden Controller zwei Testfälle, entweder „Not found“ beziehungsweise „Internal Server Error“ oder „OK“.

Vom Test der darunterliegenden Komponenten (zum Beispiel Services) wird abgesehen, da die Controller die Daten aus diesen beziehen und durch diese auch die anderen Schichten mitgetestet werden.

#### 4.3.1.1 Teacher-Controller

Es werden alle Methoden des Teacher-Controllers getestet. Hierzu gehören:

- getAllTeachers
- getTeacherById
- addTeacher

- addBulk
- editTeacher
- deleteTeacher

Bei den Methoden, die nur Daten zurückliefern und sonst keine anderen Funktionen erfüllen (getAllTeachers, getTeacherById) wird überprüft, ob der Controller die Anfrage richtig behandelt. So soll etwa ein Fehler zurückgegeben werden, falls der Datensatz nicht vorhanden und umgekehrt für einen existierenden Datensatz.

```
@Test
2 public void testGetTeacherById() throws Exception{
    MockHttpServletRequestBuilder getRequest = get("/trmanager/teacher/50000");
4    ResultActions result = mvc.perform(getRequest);
    result.andExpect(status().isNotFound());
6    getRequest = get("/trmanager/teacher/5");
    result = mvc.perform(getRequest);
8    result.andDo(print());
    result.andExpect(status().isOk());
10
}
```

Listing 4: Test der getById-Methode des Teacher-Controllers

Listing 4 auf Seite 36 zeigt den Test einer solchen idempotenten Methode. Es wird zuerst ein aus dem MockMVC-Objekt ein RequestBuilder generiert, der für die Erzeugung des Web-Requests verwendet wird. Dieser Request wird mit „mvc.perform“ abgesetzt und das Ergebnis gespeichert. Dieser Vorgang wird zwei Mal durchgeführt, einmal mit einer ungültigen ID (50000) und einmal mit einer Gültigen. Die Methode „andExpect“ ist für die Überprüfung des Status zuständig, hier verwendet mit „isNotFound“ für die ungültige ID und „isOk“ für die gültige.



```
1  @Test
   public void testAddTeacher() throws Exception{
3      Teacher t = new Teacher(null, "TESTTEACHER", "TEST", "Herr");
      Teacher t_err = new Teacher(null, "TESTTEACHER2", null, "Herr");
5
      ObjectMapper mapper = new ObjectMapper();
7      PrettyPrinter printer = new DefaultPrettyPrinter();
      ObjectWriter writer = mapper.writer().with(printer);
9      String json = writer.writeValueAsString(t);
      String json_err = writer.writeValueAsString(t_err);
11
      MockHttpServletRequestBuilder postRequest = post("/trmanager/teacher/")
13          .contentType(MediaType.APPLICATION_JSON)
          .content(json);
15      ResultActions result;
      result = mvc.perform(postRequest);
17      result.andDo(print());
      result.andExpect(status().isOk());
19
      postRequest = post("/trmanager/teacher")
21          .contentType(MediaType.APPLICATION_JSON)
          .content(json_err);
23      result = mvc.perform(postRequest);
      result.andDo(print());
25      result.andExpect(status().isInternalServerError());
   }
```

Listing 5: Test der Add-Methode des Teacher-Controllers

Analog dazu laufen die Tests für die nicht-idempotenten Methoden des Teacher-Controllers ab, hier muss aber anstatt eines GET-Requests ein POST-Request erzeugt werden

und die zu übertragenden Daten serialisiert. Dafür sind jeweils der ObjectMapper, der PrettyPrinter und der ObjectWriter zuständig, die das übergebene Objekt in JSON umwandeln. Nach der Umwandlung liegt das Objekt als String vor, das dem Request mitgegeben werden kann. Auch hier werden die zwei Testfälle behandelt, einmal mit gültigen Daten (Alle nötigen Werte sind gesetzt) und ungültigen Daten (Ein benötigter Wert wird auf null gesetzt). Der Rest des Tests ist analog zum idempotenten Test, nur dass hier anstelle eines „Not found“ ein „Internal Server Error“ geworfen wird. Für den hier getesteten Fehler würden eventuell andere Statuscodes besser passen wie zum Beispiel „Unprocessable Entity“, der dem Anwendungsentwickler mehr Aufschluss geben würde.

Die anderen Controller werden analog zu diesem Controller getestet, diese unterscheiden sich lediglich im Klassennamen, die Semantik der Tests bleibt gleich.

#### 4.3.1.2 User-Controller

Da der User-Controller nur über die Authenticate-Methode verfügt (Die Benutzer werden auf Datenbankebene eingefügt und sollen nicht geändert werden) muss nur diese getestet werden.

```
@Test
2  public void testAuthenticate() throws Exception{
    User u = repository.findOne(1);
4    User u_err = new User(1,"test","test");
    ObjectMapper mapper = new ObjectMapper();
6    PrettyPrinter printer = new DefaultPrettyPrinter();
    ObjectWriter writer = mapper.writer().with(printer);
8    String json = writer.writeValueAsString(u);
    String json_err = writer.writeValueAsString(u_err);
10   MockHttpServletRequestBuilder authRequest = post("/trmanager/auth/")
```

```
        .contentType(MediaType.APPLICATION_JSON)
12        .content(json);

    MockHttpServletRequestBuilder authRequestErr = post("/trmanager/auth/")
14        .contentType(MediaType.APPLICATION_JSON)
        .content(json_err);

16

    ResultActions result = mvc.perform(authRequest);
18    result.andDo(print());
    result.andExpect(status().isOk());

20    ResultActions result_err = mvc.perform(authRequestErr);
    result_err.andDo(print());
22    result_err.andExpect(status().isUnauthorized());
}
```

Listing 6: testAuthenticate des User-Controllers

Listing 6 zeigt die Methode zum Testen der Authenticate-Methode des User-Controllers. Zuerst werden wie üblich die Hilfsobjekte zum generieren des JSON-Strings erzeugt. Diese werden benutzt um zwei User-Objekte zu serialisieren, einmal einen validen Benutzer (wird aus der Datenbank geholt) und einen Invaliden. Der valide Benutzer sollte ein „OK“ vom Backend erhalten, der Invalide ein „NOT\_AUTHORIZED“.

## 5 Clients

Die Clients sind dafür zuständig um dem Benutzer ein Interface zu bieten, mit dem er den Webservice nutzen kann. In Zuge dieser Arbeit werden nur graphische Benutzerinterfaces betrachtet, da diese für die Benutzer einfacher zu erlernen sind und auch einen einfachen Workflow bieten. Umgesetzt wurden der native Windows-Client und ein Web-Client, der lesenden und teilweise schreibenden Zugriff (Incident anlegen) ermöglicht. Beschrieben wird der gemeinsame Teil, der die Daten versendet und empfängt. Die eigentlichen Benutzeroberflächen werden dann in eigenen Kapiteln behandelt.

### 5.1 Anforderungen

Jeder der Clients muss eine Komponente besitzen, um Daten ans Backend zu senden und von diesem zu erhalten. Dies wird im Falle der .NET-Basierten Clients über die `HttpWebRequest`-Klasse aus der `System.Net`-Bibliothek. Mithilfe dieser Methode wird der Zugriff auf das Backend abstrahiert, ähnlich einer DAO-Schicht, der `TRManager_http_client`. Die Funktionsweise dieser Klasse wird in Kapitel 5.2.2 auf Seite 43 näher erläutert.

Neben der Komponente zur Kommunikation sollen die Clients es dem Anwender ermöglichen, Vorfälle einzutragen und wieder auszutragen (entlassen). Für den administrativen Benutzer soll zudem die Möglichkeit bestehen, Datensätze zu verwalten (Im Sinne von bearbeiten, neue erstellen und löschen) und Daten zu im- und exportieren.

### 5.2 Implementierung

Jeder im Zuge dieser Arbeit implementierte Client wurde in C# mit dem .NET-Framework geschrieben, der Web-Client verwendet zusätzlich für die Web-Funktionalität

das ASP.NET-Framework. Dies bedeutet, dass ein großer Teil des Codes wiederverwendet werden kann, wie etwa die Datenmodell-Klassen und die Methoden zum empfangen und senden von Daten über das Netzwerk (respektive über HTTP). Die gemeinsame Basis der Clients wird in diesem Kapitel beschrieben.

### 5.2.1 Model

Das Model enthält die Datenstrukturen, die vom Backend vorgegeben werden. Diese werden vom Backend festgelegt und müssen in den Clients exakt umgesetzt werden. Zu den Model-Klassen gehören wie in Kapitel 4.2.1 auf Seite 27 beschrieben:

- Teacher
- Form (Schulklasse)
- Student (Schüler)
- Incident (Vorfall)
- Comment (Vorgefertigter Kommentar)

Der einzige Unterschied zum serverseitigen Modell besteht darin, dass für die Basis-Klasse „LazyObject“ (Siehe Anhang) keine Klasse implementiert wurde, da die Funktionalität, die LazyObject erfüllt (Flag zum Anzeigen ob das Objekt vollständig geladen ist und Hilfsklassen), nur auf dem Server benötigt wird und für den Client bis auf die vererbte ID uninteressant ist. Daraus ergibt sich auch die Änderung, dass die Klassen im Gegensatz zum Model im Backend die ID als direktes Attribut führen.

```
1 public class Teacher
  {
3     public int ID { get; set; }
    public String name { get; set; }
5     public String abbreviation { get; set; }
    public String salutation { get; set; }
7     public String email { get; set; }
    [JsonConstructor]
9     public Teacher(int ID, String Name, String Abbreviation,
        String Salutation, String email)
11    {
        this.ID = ID;
13        this.name = Name;
        this.abbreviation = Abbreviation;
15        this.salutation = Salutation;
        this.email = email;
17    }
  }
```

Listing 7: Teacher-Model-Klasse im Client

Listing 7 zeigt als Beispiel einen Auszug aus der Teacher-Model-Klasse des Clients. Was hierbei zu beachten ist, dass die Attribute exakt so benannt sind wie im Backend. Dies ist später für die Serialisierung und Deserialisierung der Daten mit JSON wichtig, da sonst die Attribute nicht richtig zugewiesen werden können. Zudem wird mit „[JsonConstructor]“ der Konstruktor markiert, mit dem die Objekte deserialisiert werden sollen. Das Framework für die Umwandlung von Daten im JSON-Format wird in Kapitel 5.2.2.2 auf Seite 46 erläutert.

### 5.2.2 HTTP-Client

Der HTTP-Client ist sozusagen das „Herstück“ der Clients. Er empfängt die Daten und bereitet diese für die Verwendung auf und serialisiert und sendet Objekte an das Backend. Es abstrahiert ähnlich einer DAO-Schicht die „Abfragen“ (in diesem Kontext Web-Requests) vom Programmierer und bietet eine komfortable Möglichkeit, den Web-Service zu nutzen. Der HTTP-Client benutzt Generics und lässt sich so an fast jeden Webservice anpassen.

Es wurden für die wichtigsten CRUD-Methoden (CRUD: Create, Read, Update, Delete) Methoden implementiert:

- add (POST)
- addBulk (POST)
- edit (PUT)
- delete (DELETE)
- getById (GET)
- getAll (GET)

Die grundsätzliche Funktionsweise der Methoden unterscheidet sich wieder in idempotente und nichtidempotente Methoden, dies wird an Auszügen der Methoden „getById“ und „add“ veranschaulicht:

```
public T getById(int id)
2 {
    request = WebRequest.Create(protocol+"://" + host + "/" + application_name +
4    HttpWebResponse response = (HttpWebResponse)request.GetResponse();
```

```
WebHeaderCollection header = response.Headers;
6 string response_text = "";
var encoding = ASCIIEncoding.ASCII;
8 using (var reader = new System.IO.StreamReader
    (response.GetResponseStream(), encoding))
10 {
    response_text = reader.ReadToEnd();
12 }
return JsonConvert.DeserializeObject<T>
14 (response_text, new IsoDateTimeConverter
    { DateTimeFormat = "yyyy-MM-dd hh:mm:ss" });
16 }
```

Listing 8: getById-Methode des Http-Clients

Listing 8 auf Seite 43 zeigt die getById-Methode des HTTP-Clients. Die wesentlichen Schritte bestehen im Erzeugen des Requests und dem Festlegen der Request-Parameter (Zeile 3), dem „Abholen“ des Ergebnisses (HttpWebResponse, Zeile 4) und dem Lesen der Response-Daten (reader.ReadToEnd(), Zeile 10). Der so erhaltene String wird danach per JsonConvert (Json.NET Framework, siehe Kapitel 5.2.2.2) deserialisiert und zurückgegeben. Der Rückgabewert der Methode wird festgelegt durch den Typ, der bei Erzeugung zur Typisierung mitgegeben wurde, zum Beispiel Teacher. Zudem ist zu beachten dass ausser dem Query-String keine weiteren Informationen an das Backend geschickt werden, da diese Methode nur Daten aus dem Backend holt und sonst keine weiteren Änderungen vornimmt. Die anderen Methoden, die GET verwenden sind sehr ähnlich implementiert, diese unterscheiden sich nur durch einen anderen Einstiegspunkt.

Anders sieht es bei der Add-Methode aus, hier werden per POST Daten an das Backend übermittelt:



```
public T add(T obj)
2 {
    request = WebRequest.Create(protocol+"://" + host + "/" + application_name +
4     request.Accept = "application/json";
    request.ContentType = "application/json";
6     request.Method = "POST";
    ASCIIEncoding encoding = new ASCIIEncoding();
8     Byte[] bytes = encoding.GetBytes(JsonConvert.SerializeObject(obj,
serializerSettings));
    Stream newStream = request.GetRequestStream();
10    newStream.Write(bytes, 0, bytes.Length);
    var response = request.GetResponse();
12    var stream = response.GetResponseStream();
    var sr = new StreamReader(stream);
14    var content = sr.ReadToEnd();
    return JsonConvert.DeserializeObject<T>(content);
16 }
```

Listing 9: Add-Methode des Http-Clients

Analog zur `getById`-Methode wird auch ein Request erzeugt, dieser benötigt aber weitere Informationen damit das Backend diesen versteht. Dazu gehört der „`ContentType`“ (In welcher Form die Daten geschickt werden) und „`Accept`“ (Legt fest in welchem Format geantwortet werden soll). Nun wird das übergebene Objekt serialisiert (JSON-Format) und an den Request angehängt (Zeile 10). Der fertige Request wird an den Server geschickt und die Antwort als Response gespeichert (Zeile 11). Die nun erhaltenen Daten entscheiden darüber ob die Methode erfolgreich war, dann erhält man das gesendete Objekt als Bestätigung mit ID zurück, falls nicht enthält der Response eine Server-Fehler-Nachricht (Internal Server Error) und die Methode bricht ab. Die anderen Methoden, die Daten per POST übertragen (`addBulk`, `edit`, `delete`) sind gleich aufge-

baut, sie unterscheiden sich lediglich durch einen anderen Einstiegspunkt (addBulk) oder einer anderen HTTP-Methode (PUT, DELETE).

#### 5.2.2.1 Repository-Utility

Die Repository-Utility-Klasse abstrahiert die Verwendung der einzelnen HTTP-Clients, die für jede Model-Klasse erstellt werden. Dabei bietet es Methoden zum Anlegen, Verändern und Löschen aller Model-Klassen(zum Beispiel addTeacher oder editIncident). Die Daten werden über die Methode „refreshData“ vom Backend nachgeladen. Hierfür wird der Backup-Controller verwendet, da er innerhalb eines Requests alle Daten ausliefern kann. Dies führt zu einer Reduktion der Anzahl der Requests, da der Client sich aus den schon geladenen Daten bedienen kann und nur bei Änderungen die Daten nachladen muss.

#### 5.2.2.2 Json.NET Framework

Da das .NET-Framework nicht über die Möglichkeit verfügt, Daten im JSON-Format zu serialisieren und deserialisieren, wurde für das Projekt die Bibliothek „Json.NET“ von Newtonsoft eingebunden. Es bietet die Möglichkeit, Objekte ins JSON-Format umzuwandeln und JSON-Strings zurück in Objekte. Hierfür muss lediglich die zu serialisierende/deserialisierende Klasse mit einem Flag versehen werden, die dem Framework den zu benutzenden Konstruktor anzeigt. Der Rest erfolgt automatisch, solange man die Attribut-Namen einhält, die vom Backend festgelegt wurden. Zudem besteht die Möglichkeit, weitere Konfigurationen wie zum Beispiel erweiterte Mappings und Ähnliches festzulegen.

Im einfachsten Fall erfolgt die Serialisierung über einen Aufruf an das Framework ohne weitere Einstellungen:

```
JsonConvert.SerializeObject(obj);
```

„obj“ ist hierbei das zu serialisierende Objekt. Die Deserialisierung erfolgt analog dazu mit:

```
1      JsonConvert.DeserializeObject<T>(content);
```

Der zurückzugebende Typ muss hier mit Generika festgelegt werden, im Falle des Http-Clients wird dies bei der Erzeugung festgelegt. „content“ stellt einen String im JSON-Format dar, der vom Framework geparsed und anhand des markierten Konstruktors in ein Objekt vom Typ „T“ umgewandelt wird.

Erweiterte Konfigurationen können mit einem „JsonSerializerSettings“-Objekt festgelegt werden.

```
1      new JsonSerializerSettings {  
        PreserveReferencesHandling = PreserveReferencesHandling.None,  
3      ReferenceLoopHandling = ReferenceLoopHandling.Ignore  
    };
```

Hier kann zum Beispiel die Behandlungsstrategie von Referenzschleifen (Zirkuläre Abhängigkeiten, die zu einer Endlosrekursion führt) oder wie Referenzen allgemein behandelt werden sollen festgelegt werden.

Json.NET ist Open Source-Software und wurde unter der MIT-Lizenz freigegeben und kann somit ohne Restriktionen in kommerzieller Software verwendet werden.

### 5.3 Nativer Client

Der native Client stellt die grafische Benutzeroberfläche dar, die zusätzlich auch die Programm-Logik enthält. Der Client ist als Model View Controller (MVC) Anwendung konzipiert. Die einzelnen funktionalen Anforderungen an den Client sind in Kapitel 5.1 aufgeführt. Im folgenden werden die einzelnen Aspekte und Elemente aufgeführt und erläutert.

#### 5.3.1 Design

Das Kapitel des Designs beschäftigt sich nicht nur mit dem äußerlichen Aspekt des nativen Clients. Die Workflows bilden den zweiten Abschnitt dieses Kapitels.

##### 5.3.1.1 Oberfläche

Das Hauptaugenmerk in der Gestaltung der Oberfläche liegt wie in Kapitel 2.3 auf Seite 8 erklärt in der Benutzerfreundlichkeit. Ebenfalls wichtig ist es aber auch ein ansprechendes und modernes Design zu wählen, um die User zum Arbeiten mit dem Programm zu animieren. Aus diesen Gründen wurde sich für das Material Design von Google entschieden. Dieses besticht durch sein minimalistisches Flat Design. Im Grunde wird hier auf Ränder und Linien im klassischen Sinne verzichtet. Zum Hervorheben von interaktiven Schaltflächen werden diese mit einem Schatten versehen. Nach dem Login über eine Seite welche sich auf das Wesentliche beschränkt, wird man auf eine Oberfläche mit mehreren Reitern geführt. Für den Lehrer, welcher Schüler im Trainingsraum betreut (folgend User genannt) steht ein einzelner Reiter zur Verfügung. Über diesen lassen sich alle Aufgaben des Users erledigen. Dem Administrator stehen noch weitere Reiter zur Verfügung. Unter dem Reiter „Administration“ wird sowohl der Import, als auch der Export der Datei gemanaged. Der Reiter „Einstellungen“ ermöglicht es die Serververbindung (IP, Port) anzugeben. Außerdem kann hier die Länge

der Unterrichtsstunden und der Pausen angegeben werden. Dies soll als Grundlage für eine Zeitanzeige verwendet werden. Im Reiter „Datensätze verwalten“ hat der Administrator die Möglichkeit zeitunabhängig die Informationen zu Incidents, Lehrern und Schülern zu bearbeiten.

### 5.3.2 Workflows

Die einzelnen Workflows werden im folgenden erläutert. Dabei werden sowohl die Schritte des Nutzer aufgeführt, als auch die wesentlichen technischen Hintergrundprozesse.

#### Schülerbesuch eintragen

Um einen Schülerbesuch einzutragen muss im linken Teil der Oberfläche zu nächst die Klasse des Schülers aus der Liste ausgewählt werden. Sobald man sich innerhalb der Liste befindet kann der User durch einfaches Tippen von Buchstaben in der Liste navigieren. Beispielsweise wird durch Eingeben eines Klassennamens diese direkt ausgewählt und angezeigt. Sobald eine Klasse ausgewählt wurde, füllt sich direkt daneben eine Liste mit den Schülern dieser Klasse. Um bei dem navigieren durch die Oberfläche nicht den Überblick zu verlieren wird in dem Spaltenkopf der Schülerliste die aktuell ausgewählte Klasse notiert. Im Hintergrund wertet das Programm bei jeder Veränderung der Markierung innerhalb der Liste die aktuelle Auswahl aus. Dazu wird zunächst geprüft ob überhaupt etwas markiert ist, denn bei dem Wechsel der Selektion ist zu beachten, dass diese aus den folgenden zwei Schritten besteht: Abwählen der vorherigen Selektion und Selektion der neuen Auswahl. Ist sichergestellt, dass es sich um die eigentlich Selektion handelt, wird der Code zur Anzeige der Schüler einer Klasse ausgeführt. Die wichtigsten Befehle dazu sind im Listing 10 aufgeführt:

```
private void lv_form_SelectedIndexChanged(object sender, EventArgs e)
2  {
    (...)
4      foreach (ListViewItem li in selectedItem)
        selected = (Model.Form)li.Tag;
6      studentList.Items.Clear();
        studentList.Columns.Clear();
8      if (selected != null)
        {
10         studentList.Columns.Add("Student (" + selected.name + ")",
            studentList.Width - 20);
12         foreach (Student s in selected.getStudents())
            {
14             ListViewItem stud = new ListViewItem();
                stud.Tag = s;
16             stud.Text = s.getGivenname() + "," + s.getSurname();
                studentList.Items.Add(stud);
18         }
        }
20    (...)
}
```

Listing 10: Klasse ausgewählt

Um sicherzustellen, dass sämtliche Daten der vorherigen Klasse aus der Liste entfernt werden, wird die komplette Spalte gelöscht. Zur besseren Übersicht der aktuellen Selektion wurde dem Spaltennamen der Schülerliste in Zeile 10 des Listings die Klassenbezeichnung angefügt. Durch die Abstraktion der Daten sind sämtliche Schüler dem Klassenobjekt der zugehörigen Klasse angehängt. Daher kann auf diese über das Klassen-Objekt mit einer foreach-Schleife zugegriffen werden. Die Objekte sind den Listen in dem Attribut „Tag“ hinterlegt, um diese für weitere Aktionen verwenden zu

können. Da der Tag aber nicht für den Benutzer der Anwendung sichtbar ist, wird zusätzlich zu einem Eintrag das Attribut „Name“ gesetzt. Nach Betätigung des Buttons „Vorfall anlegen“ wird ein separates Fenster geöffnet. Um dies hervorzuheben wird die MainForm leicht durchsichtig. Dies geschieht über die Methode „Deactivate“ und den Eigenschaftswert „Opacity“. In der folgenden Form erhält der Nutzer eine Liste mit sämtlichen Lehrern, ein Freitextfeld für Bemerkungen und eine Auswahl von der Standard Begründungen. Sämtliche hier enthaltenen Daten stehen in der Datenbank und werden durch die Klasse RepositoryUtility (Kapitel 5.2.2.1 Seite 46) zur Verfügung gestellt. Dies bedeutet, dass auch hier lediglich mit Listen gearbeitet wird. Durch den Button „Vorfall anlegen“ wird ein neues Incident-Objekt erstellt und mit den Daten aus dem Formular und der aktuellen Zeit gefüttert. Der Inhalt des Freitextfelds und die Auswahl der Begründung bilden dabei das Attribut „Comment“. Dieses neu erstellte Incident Object wird mittels über RepositoryUtility dem Backend übermittelt. Damit die Daten in der Hauptübersicht ebenfalls aktuell sind wird vor dem Schließen des „Create Incident“ Fensters noch ein Refresh auf dem Hauptfenster aufgerufen:

```
1 form_main.mainFormReference.refreshInterfaceData();
```

In der Refresh Methode des Hauptfenster (Kapitel 9.2 Seite 71) werden über die RepositoryUtility Klasse die aktuellen Daten aus dem Backend abgerufen und in den zuvor geleerten Listen angezeigt.

### Details zu einem Incident anzeigen

Um sich Details zu einem Incident anzeigen zu lassen, muss in der Tagesansicht selektiert werden. Sobald dieser ausgewählt ist, werden unten rechts in der Detailbox Infos zum betreffenden Incident angezeigt (Anwenderhandbuch, Kapitel 3 Abbildung 3: 2). Dazu gehören:

- Scheinnummer

- Name
- Ankunft
- Schickender Lehrer
- Kommentar

Im Hintergrund wird das Event „SelectedIndexChanged“ ausgelöst. Die Methode prüft zunächst, ob ein Element ausgewählt ist und übernimmt dann die Daten in die Detailbox.

### Schüler aus dem Trainingsraum entlassen

Um einen Schüler aus dem Trainingsraum zu entlassen muss der betreffende Incident wie in „Details zu einem Incident anzeigen“ beschrieben ausgewählt werden. Nun kann der Button „Entlassen“ betätigt werden (Benutzerhandbuch, Kapitel 3 Abbildung 3: 6). Bei Betätigung dieses Buttons wird (15 Minuten Anwesenheit vorausgesetzt) eine Infobox zur Bestätigung angezeigt und der Schüler ist entlassen. Im Hintergrund löst dies die Methode „btn\_release\_student\_Click“ aus:

```
1 if (Incident_info_box.Tag != null && Incident_info_box.Tag is Incident)
   {
3     RepositoryUtility.releaseStudent((Incident)Incident_info_box.Tag);
   }
5 this.refreshInterfaceData();
```

Das vorangestellte Listing zeigt die ausgelösten Funktionen nach dem Betätigen von „Entlassen“. Der Incident wird aus der Liste extrahiert und an die Methode „releaseStudent“ (RepositoryUtility) übergeben, die die Entlassung an das Backend weitergibt. Bei einer Entlassung wird in den Incident die Zeit des Verlassens eingetragen und ist



somit abgeschlossen. Um dem Benutzer die aktuellen Daten anzuzeigen werden die angezeigte Daten mit „refreshInterfaceData“ aktualisiert.

### Rückkehrer eintragen

Um einen Rückkehrer einzutragen, muss zunächst die Schaltfläche „Rückkehrer“ betätigt werden (7). Das sich nun öffnende Fenster zeigt eine Liste mit allen innerhalb der letzten 5 Tage eingetragenen Incidents. Aus dieser Liste kann der entsprechende Incident ausgewählt werden und über einen Klick auf den Button „Eintragen“ eingetragen werden. Die Scheinnummer bleibt hierbei die gleiche.

Im Hintergrund wird die Liste mit vergangenen Incidents aus der Liste aller Incidents gefüllt. Hierbei werden jedoch nur Incidents übernommen, die ein Entlassungs-Datum gesetzt haben und das Datum innerhalb der letzten 5 Tage liegt. Das Eintragen erfolgt technisch wie in „Schülerbesuch eintragen“, nur wird hier die Scheinnummer manuell gesetzt. Nach dem Eintragen werden die Anzeigedaten über das übliche

```
1 refreshInterfaceData();
```

aktualisiert.

### Import/Export

Um einen Import beziehungsweise Export zu starten (Beide Vorgänge sind gekoppelt) muss in den Reiter „Administration“ gewechselt werden. Die Schaltfläche „Daten importieren“ löst den Import/Export-Vorgang aus. Der Benutzer muss nun die Dateien für den Import auswählen (Schülerdaten und Lehrerdaten). Die Daten werden daraufhin im Backend gelöscht und die neuen Daten eingefügt. Im Hintergrund wird dies über den Backup-Controller des Backends gelöst, welcher alle Daten auf einmal enthält.

### Servereinstellungen ändern

Die Servereinstellungen werden im Reiter Einstellungen verändert. Durch einen Klick in die Textbox können jeweils die folgenden drei Werte bearbeitet werden:

- Serveradresse
- Port
- Endpoint

Ein roter Balken unterhalb der Textbox markiert dabei die aktuelle Position. Durch Betätigung des „Konfiguration speichern“ wird die Methode zur persistenten Speicherung der Konfiguration aufgerufen:

```
1 private void btn_save_config_Click(object sender, EventArgs e)
{
3     configuration_store.school_begin_time = txtbox_begin_time.Text;
    configuration_store.backend_entry_endpoint = txtbox_endpoint.Text.ToLower();
5     int.TryParse(txtbox_lesson_length.Text, out configuration_store.lesson_length);
    configuration_store.backend_port = txtbox_port.Text;
7     configuration_store.backend_address = txtbox_server_ip.Text;
    configuration_store.reload_config();
9     configuration_store.config_writeback();
    if (!configLoaded && RepositoryUtility.refreshData())
11     materialTabControl1.SelectedTab = materialTabControl1.TabPages[0];
    this.refreshInterfaceData();
13 }
```

Listing 11: Speicherung der Konfiguration

Diese Methode bewirkt durch ein Aufrufen des Befehls „configuration\_store.config\_writeback()“, dass die Konfiguration persistent in einer Datei geschrieben wird. Um die Komplexität dafür abzukoppeln wurden die eigentlichen Methoden zum schreiben in einer Datei in die Klasse „configuration\_store“ (Anhang 9.3 Seite 73) ausgelagert.

### **Servereinstellungen ändern (ohne Login)**

Falls der Client beim Starten die Konfigurationsdatei nicht lesen kann, hat der Anwender die Möglichkeit, diese wie im vorangegangenen Abschnitt zu ändern, ohne sich vorher zu authentifizieren

### **Klasse ändern**

Um eine Klasse zu ändern muss zuerst in den Reiter „Datensätze verwalten“ gewechselt werden. Über die Schaltflächen Klasse, Lehrer Schüler und Vorfall erreicht man die Listen der verschiedenen Elemente. Im Falle der Klasse wäre dies A. Man wählt den gewünschten Datensatz aus und betätigt die Schaltfläche „Verändern“. Im sich öffnenden Fenster kann man die Daten ändern und bestätigen. Im Hintergrund wird das Klassenobjekt mit den geänderten Daten an das Backend geschickt (Über RepositoryUtility.editForm). Falls das Backend die Änderung bestätigt, bekommt der Benutzer eine Benachrichtigung über den Erfolg der Änderung.

### **Klasse hinzufügen**

Um eine Klasse hinzuzufügen muss wie in „Klasse ändern“ in den Reiter „Datensätze verwalten“ gewechselt werden. Anstelle des Buttons „Verändern“ wird nun die Schaltfläche „Hinzufügen“ ausgewählt. Hierfür muss kein Eintrag in der Liste ausgewählt werden. Im sich geöffneten Fenster werden nun die Werte der Klasse eingetragen und mit „Hinzufügen“ bestätigt.

Aus den eingegebenen Werten wird ein Klassenobjekt erstellt und an das Backend geschickt (Über RepositoryUtility.addForm). Bei Erfolg erhält der Client das Objekt mit der vom Backend vergebenen ID zurück und der Benutzer bekommt eine Bestätigung.

**Klasse löschen**

Um eine Klasse zu löschen, muss wie in „Klasse ändern“ in den Reiter „Datensätze verwalten“ gewechselt werden. Die gewünschte Klasse wird aus der Liste ausgewählt und betätigt die Schaltfläche „Löschen“. Diese Anfrage muss noch einmal bestätigt werden, dann wird das Objekt gelöscht.

Im Hintergrund läuft dies über die deleteForm-Methode aus RepositoryUtility. Das Objekt wird an das Backend geschickt, dieses schickt ein „OK“ zurück falls der Vorgang erfolgreich war.

**Änderung anderer Datentypen**

Die Änderungen der anderen Datentypen funktioniert analog zu der beschriebenen Methode für die Klassen.

## 5.4 Web Client

Der Web-Client soll über den Browser erreichbar sein und den gleichen Funktionsumfang wie der native Client bieten. Um aber den Zeitrahmen dieser Arbeit nicht zu sprengen wurde der Web-Client nur exemplarisch mit einem geringen Funktionsumfang implementiert

### 5.4.1 Funktionsumfang

Der Web-Client sollte in etwa dem Funktionsumfang des native Clients entsprechen, um einen mobilen Zugriff auf die Anwendung zu ermöglichen. Dazu zählen alle Funktionen, die für den nativen Client gelistet wurden, wie zum Beispiel Incidents anlegen, Datensätze verwalten und andere.

Der eigentliche Funktionsumfang des Web-Clients beschränkt sich auf das Anzeigen der Klassen sowie eine List mit allen aktuellen Incidents inklusive Zeitzähler.

### 5.4.2 Design

Das Design wurde mit ASP.NET und Bootstrap umgesetzt und an das Design des nativen Clients angelehnt.

Students

BFG115

Hamdy	Aden
Gurhan	Ahmed
Yasmin	Akchich
Samar	Al-Malliji
Michelle	Dalmus
Marlene	de Sa
OEzge	Eris
Saskia	Fingerhuth
Svenja	Hagen
Ravide	Halidi
Isabell	Harder
Bedran	Jawich
Nagihan	Karaduman
Antonia	Markovic
Jana	Meler
Nicole	Mitulla
Sidra	Mohammad
Lena	Pisano
Nadine	Plag
Janina	Puetz
Janine	Rohrmann
Rabia	Saleh
Lia	Staudinger
Marina	Stollenwerk
Betuel-Tugce	Uguz

ID	Scheinnummer	Ankunft	Timer	Kommentar
1	1	2016-06-22 10:23:04	02:57:38	trinkt, isst oder kaut Kaugummi w??hrend des Unterrichts

Vorfall hinzufügen

Abbildung 4: Design des Webclients

Abbildung 4 zeigt das grundlegende Design des Webclients. Links wird eine Liste mit Klassen und Schülern angezeigt und in der Mitte die Vorfallsliste.

Die Vorfallsliste wurde technisch mit einer GridView umgesetzt, die dynamisch Daten nachladen kann ohne die gesamte Seite zu aktualisieren. So können per AJAX Daten nachgeladen werden, ohne dass der Benutzer davon etwas merkt. Anderweitige Funktionen wie das Anlegen eines Incidents oder das Entlassen von Schülern aus dem Trainingsraum wurden bis jetzt nicht umgesetzt, sind aber mit den vorhandenen Funktionen einfach umsetzbar.

## 6 Fazit

Im Laufe des Projektes wurden die User-Stories durchgearbeitet. Dabei wurden einige umgesetzt, andere neu bewertet und weitere auch gänzlich gestrichen. Folgend werden die wichtigsten User-Stories und deren Bearbeitung dargestellt.

Die wesentlichen Grundfunktionen mit der Priorität eins wurden im Rahmen dieses Projektes vollständig umgesetzt. Die Anforderungen zu Priorität Zwei wurden im wesentlichen mit kleineren Änderungen ebenfalls umgesetzt. Die druckfertige Nachricht nach dem dritten bzw. den fünften Trainingsraum-Besuch ist bisher noch nicht umgesetzt. Die Autovervollständigung zur vereinfachten Suche von Schülern wurde durch die vorgeschaltete Auswahl der Klasse umgangen. Die Autovervollständigung sollte die Problematik von komplizierten und auf Papier schwer lesbaren Name beheben. Die gewählte Vorgehensweise erfüllt diesen Zweck ebenfalls und dient zudem auch der Übersicht in der Oberfläche. Die Anforderungen mit der Priorität Drei bis Fünf wurden noch nicht umgesetzt und müssen noch geprüft werden. Zu diesen Anforderungen gehört im wesentlichen die Verwendung von Mailadressen. Um diese Funktionen umzusetzen müssen mehrere Bedingungen erfüllt sein. Zum einem werden die Mailadressen der Lehrer benötigt und zum anderen muss ein Server zur Verfügung gestellt werden, welcher die Mails an die Lehrer verschicken kann. Das Verwaltungstool SchILD-NRW, aus welchem sämtliche Daten exportiert werden speichert nicht die Mailadressen der Lehrer. Außerdem steht dem Berufskolleg mit dem kooperiert wird weder die Mittel noch die Kompetenzen für einen eigenen Mail-Server zur Verfügung.

Die Statistiken und das Archiv wurden ebenfalls noch nicht umgesetzt. Die Vorbereitungen, wie etwa der Export der Daten wurden sind allerdings schon implementiert.

Die unterstützende Funktion des externen Zugriffs wurde mittels des Web-Clients realisiert. Weiteres dazu in Kapitel 2.7.3 auf Seite 11. Zu Beginn des Projekts gab

es noch die Anforderung, dass jeder angemeldete Lehrer die Möglichkeit haben soll einen Schüler anzulegen, sofern er in der Klasse nicht zu finden ist. Diese Anforderung entspringt der Tatsache, dass gerade zu Beginn des Schuljahres Schüler nachgepflegt werden müssen. Da es aber in Rahmen der Testphasen häufiger dazu kam, dass Lehrer bereits vorhandene Schüler eingetragen habe, weil sie in der falschen Klasse gesucht haben, wurde in Absprache mit der kooperierenden Schule beschlossen, diese Funktion wieder abzuschalten und auch aus den User-Stories zu entfernen. Ist nun ein Schüler nicht im System notiert, wird der komplette eintragen auf Papier festgehalten und vom Administrator nachgepflegt.

Das gesamte Projekt besteht wie bereits häufiger erläutert aus drei Teilen, welche alle separat installiert werden müssen. Zum Zeitpunkt des Projektabschlusses steht kein Installationsprogramm zur Verfügung. Daher wird dem kooperativen Berufskolleg bei Auslieferung die vollständige Applikation installiert und eingerichtet. Die Installation soll in Zukunft vereinfacht werden, damit diese ohne einen Entwickler durchgeführt werden kann.



## 7 Ausblick

Im Rahmen dieser Arbeit konnten bereits viele Features umgesetzt werden. Dennoch stehen noch einige Features aus. Diese sind dabei keine Grundfunktionen, sondern Funktionen, die immer wieder von den Nutzern gewünscht wurden, oder aber die Entwickler für sinnvoll erachten. Außerdem ermöglicht das Produkt eine wirtschaftliche Betrachtung, die in Kapitel 7.2 auf Seite 62 genauer erläutert wird.

### 7.1 Technische Komponenten

Auf technischer Seite gibt es einige Aspekte die im Anschluss an das Projekt noch weiter ausgearbeitet werden. Die Hauptpunkte werden in den folgenden Abschnitten vorgestellt.

#### 7.1.1 Web-Applikation

Der Webclient hat bisher nur einen sehr geringen Funktionsumfang, dieser ermöglicht es sich die im Backend vorhandenen Klassen (inklusive Schülern) und Incidents mit Zeitzähler anzuzeigen. In Zukunft soll dieser Client einen ähnlichen Funktionsumfang wie der native Client aufweisen. Dazu sollten allerdings weitere Sicherheitsaspekte betrachtet werden.

#### 7.1.2 Mobile Applikation

Zu Beginn der Arbeit wurde bereits über eine mobile Applikation nachgedacht. Durch die Web-Applikation rückte diese in den Hintergrund. nichtsdestotrotz stellt eine App für Tablets und Smartphones ein attraktives Angebot für moderne Schulen dar. Hier besteht demnach Potenzial für weitere Entwicklungen.

### 7.1.3 Statistik-Tools

Für die Auswertung der Daten bietet sich ein eigenes Statistik-Tool an. Eine solche Anforderung taucht auch schon in den User-Stories auf. Zwar ist über den Reiter Datensatzverwaltung eine Liste aller Einträge abrufbar, aber als Statistik geht diese wohl nicht durch. Hier ist eine separate Anwendung oder entsprechende Erweiterungen anzugehen.

### 7.1.4 Backend-Erweiterungen

Als weiterführende Maßnahmen wären die Einführung von Verschlüsselung sowie eine serverseitige Authentifizierung denkbar. Hierbei könnte auf Transportebene TLS eingesetzt werden, dies ist ohne großen Aufwand möglich, da dies für HTTP schon weitreichend entwickelt ist. Für die Authentifizierung gäbe es mehrere Möglichkeiten. Neben einer selbstentwickelten Authentifizierung könnte Spring Security für diese Zwecke angepasst werden. Da die Anwendung aber in einem abgeschlossenen Netz läuft wird dies eher als zweitrangig gesehen.

## 7.2 Wirtschaftlicher Faktor

Das Projekt hat als Endergebnis ein solide funktionierendes Programm, welches auf dem aktuellen Markt keine Konkurrenz hat. Zudem wird das Konzept des Trainingsraumes immer mehr auch durch die Schulministerien der Länder propagiert. Eine Vermarktung dieser Anwendung scheint daher der nächst logische Schritt. Bevor dieser Schritt angegangen werden kann, sollte das Programm zunächst durch ein dokumentiertes Pilotverfahren gebracht werden. Hier steht den Entwicklern das Robert-Wetzlar-Berufskolleg zur Seite. Dieses verwendet bereits frühere Versionen des Programmes und hat das Trainingsraum-Konzept in den Schulalltag bereits vollständig integriert.

Um das Programm zu vermarkten muss natürlich zum einen Marketing stattfinden und zum anderen die Lizenzierung und damit auch Kosten für die Kunden.

### 7.2.1 Marketing

Das Trainingskonzept wird nicht nur von den Schulministerien der Länder beworben sondern auch von unterschiedlichen Sozialpädagogen. So zum Beispiel auch Dr. Heidrun Bründel, Dipl.-Psych., und Erika Simon, Lehrerin. Zusammen haben die beiden das Buch „Die Trainingsraum-Methode“ herausgebracht. In diesem beschreiben sie unter anderem in einem Kapitel die Bürokratie der Methode <sup>7</sup>. Daraus bieten sich den Entwicklern dieser Anwendung nun zwei Wege des Marketing. Für ein großflächiges Marketing könnte mit den beiden Parteien (Buchautoren und Schulministerium) Kontakt aufgenommen werden um eine Nennung des Programmes auf deren Webseiten anzustreben. Dazu sollten aber neben dem Anwenderhandbuch (siehe Anhang 9.5) auch entsprechendes Präsentationsmaterial und mindestens eine Testumgebung des Programmes zur Verfügung stehen sollte. Eine solche Testumgebung steht bereits zur Verfügung und ist über entsprechende Verbindungsdaten und Authentifizierungsdaten erreichbar. Neben diesem Weg ist natürlich auch das direkte Anschreiben von Schulen eine Möglichkeit und sollte zukünftig in Betracht gezogen werden. Der persönliche Kontakt zu Lehrern und Dozenten von Lehrerschulungen sollte den Prozess der Verbreitung ebenfalls positiv beeinflussen.

### 7.2.2 Lizenzierung

Sobald man sich mit der Vermarktung eines Produktes beschäftigt, kommt zwangsläufig die Frage nach der Lizenzierung. Bevor man sich entscheiden kann oder auch nur

---

<sup>7</sup> Erika Simon Heidrun Bründel. *Die Trainingsraum-Methode: [Unterrichtsstörungen - klare Regeln, klare Konsequenzen.]*. Beltz, 3., erweiterte und aktualisierte Aufl. edition, 2013

mit den unterschiedlichen Lizenzen auseinander setzen kann sollten die eigenen Anforderungen klar definiert sein. Die Entwickler dieses Projektes haben dabei die folgenden Rahmenbedingungen herausgearbeitet

- Quelloffen
- Namensnennung
- keine Manipulation des Codes
- keine (kommerzielle) Wiederverwendung des Codes
- kein Vertrieb der Software durch Dritte

Die Rahmenbedingungen müssen alle evaluiert und auf ihre Tauglichkeit geprüft werden. Des Weiteren ist unter den vielen unterschiedlichen möglichen Lizenzmodellen und Lizenzarten das passende zu wählen und ggf Abstriche zu machen.

## Literatur

- [1] Erika Simon Heidrun Bründel. *Die Trainingsraum-Methode: [Unterrichtsstörungen - klare Regeln, klare Konsequenzen.]*. Beltz, 3., erweiterte und aktualisierte aufl. edition, 2013.
- [2] Steve Krug. *Dont make me think, revisited: a common sense approach to Web usability*. New Riders, 2014.
- [3] Jan Machacek et al. *Pro Spring 2.5*. Apress, Berkeley, Calif., 2008.
- [4] Dave Minter et al. *Beginning Hibernate*. Apress, New York, 3rd ed (online-ausg.) edition, 2006.
- [5] Mathias Rohr. *Sicherheit von Webanwendungen in der Praxis*. Springer Vieweg, 2015.
- [6] Bernhard Steppan. *Einstieg in Java 7: [eine umfassende und professionelle Einführung; mit vielen Beispielen und kommentierten Lösungen; Programmierung von GUIs, Datenbanken, dynamischen Websites u.v.m.]*. Galileo Press, Bonn, 4., aktualisierte aufl. edition, 2012.

## 8 Glossar

### **Android**

Betriebssystem und Softwareplattform für mobile Geräte, die von der Open Handset Alliance(gegründet von Google) entwickelt wird. Basiert auf dem Linux-Kernel und wird quelloffen entwickelt.

### **idempotent**

Als idempotent bezeichnet man Arbeitsgänge, die immer zu den gleichen Ergebnissen führen, unabhängig davon, wie oft sie mit den gleichen Daten wiederholt werden. Idempotente Arbeitsgänge können zufällig oder absichtlich wiederholt werden, ohne dass sie nachteilige Auswirkungen auf den Computer haben.

### **iOS**

IOS ist ein von Apple entwickeltes mobiles Betriebssystem. Anders als bei den Konkurrenzprodukten wie etwa Android wird iOS ausschließlich auf Apple Produkten verwendet.

### **Laufzettel**

Jeder Schüler der in den Trainingsraum geschickt wird erhält von seinem schickenden Lehrer einen Laufzettel. Auf diesem steht die Art der Unterrichtsstörung, der Name des Schülers und seine Klasse drauf. Von dem betreuenden Lehrer im Trainingsraum wird zusätzlich die vom Programm generierte Scheinnummer hinzugefügt. Dieser Laufzettel stammt aus der analogen Verwaltung des Trainingsraumes

### **Überweisungsschein**

Der Überweisungsschein ist ein anderer Begriff für Laufzettel und wird äquivalent verwendet.

## 9 Anhang

### 9.1 LazyObject

```
1  @MappedSuperclass
   public abstract class LazyObject {
3
   @JsonProperty("ID")
5   @Id
   @GeneratedValue(strategy=GenerationType.AUTO)
7   @Column(name = "id", unique = true, nullable = false)
   private Integer ID = null;
9
   @JsonIgnore
11  @Transient
   private boolean isLoading;
13
   @JsonCreator
15  public LazyObject(@JsonProperty("ID") Integer ID) {
       setId(ID);
17  }

19  @JsonProperty("ID")
   public int getId() {
21      return ID;
   }
23
   @JsonProperty("ID")
25  public void setId(Integer id) {
       if (id == null) {
```



```
27         this.ID = -1;
        } else {
29         this.ID = id;
        }
31     }

33     protected LazyObject() {
        ID = null;
35     }

37     @JsonIgnore
    public boolean getIsLoaded() {
39         return isLoaded;
    }

41
    @JsonIgnore
43     public void setIsLoaded(boolean isLoaded) {
        this.isLoaded = isLoaded;
45     }

47     @Override
    public boolean equals(Object obj) {
49         if (obj == null) {
            return false;
51         }
        if (obj == this) {
53             return true;
        }
55         if (!(obj instanceof LazyObject)) {
            return false;
57         }
```

```
        LazyObject that = (LazyObject) obj;
59     return this.ID == that.ID;
    }

61

    @Override
63     public int hashCode() {
        return super.hashCode();
65     }
}
```

## 9.2 refreshInterfaceData

```
public void refreshInterfaceData()
2 {
    if (RepositoryUtility.refreshData())
4     {
        lv_form.Items.Clear();
        lv_form.Columns.Clear();
        lv_form.Columns.Add("Klasse");
8     foreach (Model.Form f in RepositoryUtility.getForms())
        {
10         ListViewItem name = new ListViewItem();
            name.Tag = f;
12         name.Text = f.name;
            lv_form.Items.Add(name);
14     }
        lv_form.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);
16     //Incident Liste füllen
        List<Incident> all_incidents = RepositoryUtility.getIncidents();
18     lw_cur_Incident.Items.Clear();
        foreach (Incident i in all_incidents)
20     {
            if (i.isCurrent())
22     {
                ListViewItem inc = new ListViewItem();
                inc.SubItems.Clear();
24                 inc.SubItems.Add(i.ticket_ID.ToString());
                inc.SubItems.Add(i.arrival);
26                 inc.SubItems.Add(i.arrival + 15);
                String x = i.student.givenname + " " + i.student.surname;
28                 inc.SubItems.Add(x);
```

```
30         inc.Tag = i;
           lw_cur_Incident.Items.Add(inc);
32     }
       }
34     lw_cur_Incident.AutoSizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize)
       lw_cur_Incident.Columns[0].Width = 0;
36     lw_cur_Incident.Columns[1].Width = 100;
       lw_cur_Incident.Columns[2].Width = 170;
38     lw_cur_Incident.Columns[3].Width = 170;
       lw_cur_Incident.Columns[4].Width = 200;
40
       }
42     else
       {
44         MessageBox.Show("Error reading data from backend");
       }
46 }
```

### 9.3 configurationStore

```
using System;
2 using System.Collections.Generic;
using System.IO;
4 using System.Linq;
using System.Text;
6 using System.Threading.Tasks;
using TRManager_new_Client.Model;

8
namespace TRManager_new_Client
10 {
    class configuration_store
12     {
        public static Teacher logged_user;

14
        public static string backend_address="";
        public static string backend_port="";
16        public static string backend_entry_endpoint="";

18
        public static string school_begin_time="";
        public static int lesson_length=0;
20        public static List<school_break> breaks = new List<school_break>();

22
        public static String dateTimeFormat = "yyyy-MM-dd HH:mm:ss";
        public static String dateTimeFormatShort = @"hh\:mm\:ss";

24
        public static int returnee_age_threshold = 7;

26
        public static bool read_config(String path)
28        {
```

```
30         StreamReader config_file = new StreamReader(path);
        List<String> lines = Utility.readDataFromFile(config_file);
32
        foreach(String line in lines)
34     {
            String[] x = line.Split('=');
36         switch (x[0])
            {
38             case "backend_address": backend_address = x[1];
                    break;
40             case "backend_port": backend_port = x[1];
                    break;
42             case "backend_entry_endpoint": backend_entry_endpoint = x[1];
                    break;
44             case "break": callAddBreak(x[1]);
                    break;
46             case "school_begin_time": school_begin_time = x[1];
                    break;
48             case "lesson_length": int.TryParse(x[1], out lesson_length);
                    break;
50             default:
                    break;
52         }

54     }

56     return false;
    }
58     public static void callAddBreak(String conf)
    {
60
```

```
    }
62     public static void addBreak(int before, int after, int length)
    {
64         breaks.Add(new school_break(before, after, length));
    }

66
    public static void config_writeback()
68     {
        int conf_line_cnt = breaks.Count;
70         String[] conf_lines = new String[5 + conf_line_cnt];

72         conf_lines[0] = "backend_address=" + backend_address;
        conf_lines[1] = "backend_port=" + backend_port;
74         conf_lines[2] = "backend_entry_endpoint=" + backend_entry_endpoint;
        conf_lines[3] = "school_begin_time=" + school_begin_time;
76         conf_lines[4] = "lesson_length=" + lesson_length.ToString();
        int cur = 5;
78         foreach (school_break s in breaks)
        {
80             conf_lines[cur] = s.ToString();
            cur++;
82         }
        if(File.Exists("trmanager.conf"))File.Delete("trmanager.conf");
84         File.WriteAllLines("trmanager.conf", conf_lines);
    }

86     public static void reload_config()
    {
88         RepositoryUtility.incidentRepository = new TRManager_http_client<Incident>()
        RepositoryUtility.formRepository = new TRManager_http_client<Model.Fo
69         RepositoryUtility.studentRepository = new TRManager_http_client<Student>()
        RepositoryUtility.teacherRepository = new TRManager_http_client<Teacher>()
```

```
92         RepositoryUtility.commentRepository = new TRManager_http_client<Comme  
        }  
94     }  
    }
```



## 9.4 User-Stories

### User-Stories Grundfunktionen

- Import und Export der Daten / Datensicherung (Priorität 1)
  - Zu Beginn jedes Schuljahres werden die Schülerdaten aus der externen SchILD-Datenbank (Verwaltungsnetzwerk) bezogen und im CSV-Format in das Programm (Schülernetzwerk) geladen.
  - Vorher sollen alle Daten, die das Programm im letzten Schuljahr erfasst hat, zunächst exportiert und dann im Programm selbst gelöscht werden.
  - Der Datentransfer (Import/Export) erfolgt über die USB-Schnittstelle.
  - Ein Export der vom Programm erfassten Daten soll zu jeder Zeit möglich sein. Dabei sollen bei aktiven Filtern / Sortierungen (s. o.) auch nur die ausgewählten Daten exportiert werden können.
  - Zusätzlich soll eine Datensicherungsfunktion implementiert werden, die es ermöglicht, alle Daten direkt mit Datumsangabe auf einen externen Datenträger (USB) zu speichern.
  - Diese Funktionen sollen ausschließlich dem Administrator zur Verfügung stehen.
- Login Daten (Priorität 1)
  - Jeder Lehrer soll sich im Programm mit seinem Kürzel (ohne Passwort) anmelden können, um die Informationen des Laufzettels in die grafische Oberfläche eintragen zu können.
- Besuch eines Schülers eintragen (Priorität 1)

- Sobald ein Schüler den Trainingsraum betritt, wird zunächst zwischen einem neuen Vorfall und einen Rückkehrer unterschieden. Dies geschieht unabhängig vom Programm durch den betreuenden Lehrer.
- Für Rückkehrer siehe „Rückkehrer“
- Bei einem neuen Vorfall wird ein neuer Incident über ein Menü eingegeben. Dabei werden folgende Daten gespeichert:
  - \* Name, Klasse des Schülers, schickender Lehrer und eine Bemerkung zu dem Vorfall
  - \* Bei der Bemerkung soll zusätzlich zu einem Freitextfeld auch eine Auswahlliste mit Standardvorgehen bereitgestellt werden.
- Ist der Vorfall eingetragen wird eine fortlaufende Überweisungsscheinnummer angezeigt, welche vom betreuenden TR-Lehrer auf den Überweisungsschein eingetragen wird. Diese wird im Incident ebenfalls gespeichert, aber nicht durch den User ausgewählt.
- Vervollständigung (Priorität 1)
  - Bei Eingabe der Namen sollen Vorschläge zur Vervollständigung angezeigt werden. Bei keinem Vorschlag soll die Neuanlage eines Schülers erfolgen können.
  - Zur Erleichterung der korrekten Vervollständigung soll zuerst die Klassenbezeichnung des Schülers eingegeben werden können, um somit die Vervollständigungsvorschläge zu minimieren und den korrekten Schüler vorgeschlagen zu bekommen.
- Behandlungen bei drei bzw. fünf Besuchen (Priorität 1)

- Nach Hinzufügen eines Trainingsraumbesuches soll ersichtlich werden, wie oft der Schüler in dem jeweiligen Schuljahr bereits im Trainingsraum war.
- Beim dritten bzw. fünften Besuch soll als Popup ersichtlich werden, dass ein Elterngespräch bzw. eine Konferenz notwendig ist.
- Behandlung Rückkehrer TR aufgrund von Ablehnung Rückkehrplan (Priorität 1)
  - Kommt ein Schüler ohne neuen Überweisungsschein zurück in den TR, muss dies vermerkt werden. Dieser Besuch ist KEIN neuer TR-Besuch. (Rückkehrer-Button)
  - Gleiche Scheinnummer, neue Incidentnummer
- Administrationsrechte (Priorität 1)
  - Es soll ein Administrator-Account mit Passwort angelegt werden, der als alleinige Instanz die Möglichkeit haben soll, zeitlich unabhängiges Hinzufügen und Verändern bereits eingetragener Daten durchzuführen.
- Schülerdatenabruf / Information über Besuche im Trainingsraum (TR) (Priorität 2)
  - Es sollen Schülerdaten (Vor-, Nachname, Id, Klasse und Klassenlehrer), sowie Informationen über Besuche im Trainingsraum (Datum, Uhrzeit, Lehrer, Anzahl der Besuche) abgerufen und in einer grafischen Oberfläche auf dem Bildschirm ausgegeben werden.
  - Alle Daten (inkl. Lehrer, Klassen) sollen gefiltert und sortiert werden können.
  - Nur zugänglich für den Admin
- Zeitzähler (Priorität 2)

- Es soll ein Zeitzähler realisiert werden, der die Aufenthaltszeit jedes Schülers, beginnend bei 0 Minuten hochzählt und vor Erreichen von 15 Minuten mit einer roten Markierung hinterlegt ist. Bei Erreichen von 15 Minuten soll die Hinterlegung von rot auf grün umspringen und ein Button realisiert werden, mit dem die betreuende Lehrkraft den Schüler (abhängig von seinem Rückkehrplan) aus dem Trainingsraum entlassen und aus der Liste entfernen kann.
- Der Zähler soll solange weiterlaufen, bis der Button gedrückt wurde und der Schüler den Raum verlässt.
- Kehrt der Schüler aufgrund einer Rückkehrplan-Ablehnung zurück, beginnen neue 15 Minuten, was über einen Rückkehrer-Button realisiert werden soll.
- Wenn das Programm abstürzt, sollen vorhandene Daten von aktuellen Trainingsraumbesuchern nicht verloren gehen. Wird das Programm nach einem Absturz wieder gestartet, sollen diese Daten aktualisiert vorhanden sein.
- Druckfertige Nachricht (Priorität 4)
  - Ist ein Elterngespräch oder eine Konferenz notwendig, soll eine druckfertige Nachricht mit unveränderbarem Template erstellt werden können.
  - Das Template für diese Nachricht soll abhängig von der Schule gestaltet werden können
- Lehrer mit hinterlegter Mailadresse (Priorität 4)
  - Neue Lehrer sollen nur vom Administrator angelegt werden können, wobei hier ebenfalls eine Emailadresse hinterlegt werden soll.
- Mailfunktion (Priorität 5)

- Es soll eine Emailfunktion vorbereitet werden, die eine Info an die beteiligten Lehrer versendet, sobald ein Schüler den TR zum dritten, bzw. fünften Mal betritt.

#### User Stories Unterstützungsfunktionen

- Statistiken anhand der Exportdaten (Priorität 4)
  - Anhand der exportierten Daten sollen lokal Statistiken erstellt werden können (Filter- und Sortieroptionen).
- Externer Zugriff (Priorität 5)
  - Für die Administration soll ein externer Zugriff auf die Daten über eine zusätzliche Software Komponente möglich sein.
  - Diese Software ist durch zusätzliche Sicherheitsmechanismen zu schützen.

## 9.5 Anwenderhandbuch



# TRManager

Anwenderhandbuch

Version: 2.1

Florian Baier, Alexandra Neffgen

---

# Inhaltsverzeichnis

1	Einleitung .....	3
1.1	Ziel und Funktion des TRManagers.....	3
1.2	Geschlechterneutrale Formulierung .....	3
1.3	Allgemeiner Hinweis .....	3
2	Vor dem Start.....	4
2.1	Systemvoraussetzungen .....	4
3	Bedienung .....	5
3.1	Anmeldung .....	5
3.1.1	Login als Lehrer .....	5
3.1.2	Login als Admin .....	5
3.2	Hauptfenster.....	6
3.2.1	Schüler eintragen.....	7
3.2.2	Schüler entlassen .....	7
3.2.3	Rückkehrer .....	7
3.3	Adminfunktionen.....	7
3.3.1	Import .....	8
3.3.2	Export .....	8
3.3.3	Einstellungen .....	8
3.3.4	Datensätze verwalten .....	9



## **1 Einleitung**

### **1.1 Ziel und Funktion des TRManagers**

Der TRManager wurde mit dem Ziel entwickelt, die bisher ausschließlich manuell durchgeführte Verwaltung des Trainingsraumes am Robert-Wetzlar-Berufskolleg technisch zu unterstützen und die Handhabung sowie Dokumentation relevanter Daten zu erleichtern.

Folgende Funktionen stellt der TRManager jeder betreuenden Lehrkraft im Trainingsraum zur Verfügung:

- Elektronische Erfassung und Speicherung
  - der Laufzetteldaten bei Ankunft eines Schülers im Trainingsraum
  - der Entlassung eines Schülers (nach Ablauf der Mindestaufenthaltszeit und Fertigstellung des Rückkehrplans) aus dem Trainingsraum
- Automatische Benachrichtigung bei 3 bzw. 5 Trainingsraumbesuchen
- Tagesansicht mit Zeitanzeige und weiteren Informationen zu jedem Aufenthalt
- Rückkehrerfunktion zur Wiederaufnahme eines Schülers in den Trainingsraum nach Ablehnung seines Rückkehrplans

Das Programm enthält außerdem einen Administrationsmodus-Modus, welcher dazu befugt, Daten im Bedarfsfall nachträglich zu ändern. Zusätzlich stehen dem Administrator der Import neuer Schülerdaten sowie der Export bestehender, durch das Programm erstellter Daten zur Verfügung.

### **1.2 Geschlechterneutrale Formulierung**

Auf geschlechtsneutrale Formulierungen wurde aus Gründen der Lesbarkeit verzichtet. Im Text sind immer beiderlei Geschlechter gemeint. Trotzdem wurde (ohne Anspruch auf Vollständigkeit) auf eine möglichst geschlechtsneutrale Formulierung geachtet.

### **1.3 Allgemeiner Hinweis**

Im Folgenden wird davon ausgegangen, dass das Konzept des Trainingsraum bekannt ist. Ein Besuch eines Schülers im Trainingsraum wird als Vorfall bezeichnet.

## **2 Vor dem Start**

### **2.1 Systemvoraussetzungen**

Um den TRManager fehlerfrei ausführen zu können wird das Betriebssystem Microsoft Windows 7 benötigt / empfohlen.

Benötigt wird außerdem die neueste Version der Laufzeitumgebung Microsoft .NET Framework und der 2007 Office System-Treiber: Datenkonnektivitätskomponenten.

Für das Backend der Software wird sowohl Java 1.8 vorausgesetzt, als auch eine Installation von MySQL (min Version 5.7.10).

## 3 Bedienung

### 3.1 Anmeldung

#### 3.1.1 Login als Lehrer

Nachdem das Programm gestartet wurde erscheint ein Anmeldefenster. Hier kann man sich mit einem gültigen Lehrerkürzel anmelden. Dazu einfach das Kürzel als Benutzer angeben und auf „ANMELDEN“ klicken.

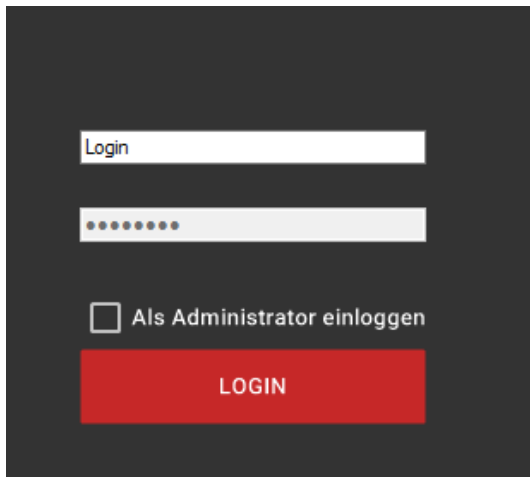
The image shows a login interface with a dark background. It features a white text input field at the top with the placeholder text "Login". Below it is a password field represented by a series of dots. Under the password field is a checkbox labeled "Als Administrator einloggen". At the bottom is a prominent red button with the white text "LOGIN".

Abbildung 1: Anmeldemaske

#### 3.1.2 Login als Admin

Um sich als Administrator anzumelden muss im Anmeldefenster der Haken „ALS ADMINISTRATOR EINLOGGEN“ gesetzt sein. Ist dieser gesetzt, wird lediglich das Passwort benötigt. Das Benutzerfeld ist für die Anmeldung als Administrator nicht von Relevanz.

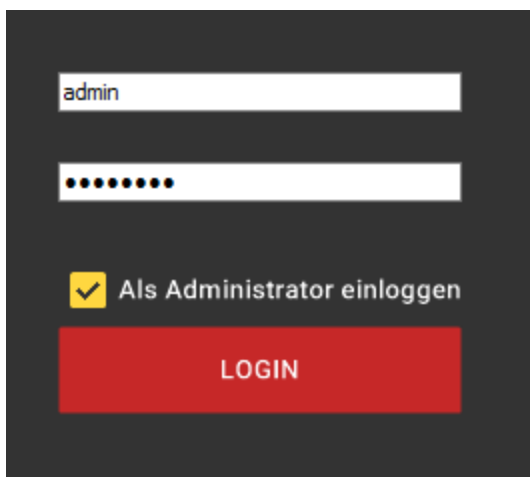
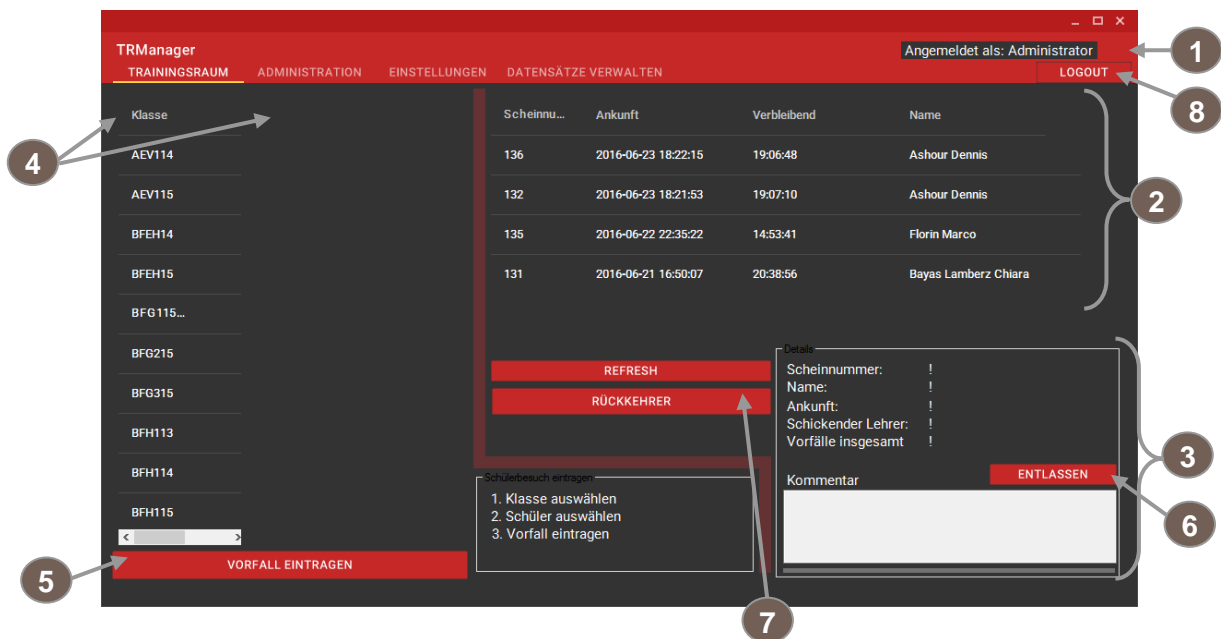
The image shows the same login interface as in the previous screenshot, but with the checkbox "Als Administrator einloggen" checked, indicated by a yellow checkmark icon. The text input field now contains the word "admin". The red "LOGIN" button remains at the bottom.

Abbildung 2: Anmeldemaske (Admin)

### 3.2 Hauptfenster

Hat man sich angemeldet, öffnet sich das Hauptfenster des TRManagers:



**Abbildung 3: Hauptfenster**

Rechts oben ( 1 ) ist zu erkennen mit welchem Benutzer man gerade angemeldet ist. Die Tagesansicht ( 2 ) enthält eine Liste der Vorfälle mit folgenden Informationen:

- Die Scheinnummer
- Die Ankunftszeit
- Die aktuelle Aufenthaltszeit des Schülers, beginnend bei 0:00 Minuten
- Den vollständigen Namen des Schülers

Die Detailbox ( 3 ) enthält zusätzliche Infos zu einem Vorfall. Der angezeigte Vorfall ist der in der Tagesübersicht als letztes ausgewählte Eintrag. Hier sind die folgenden Informationen aufgeführt:

- Die Scheinnummer
- Den vollständigen Namen des Schülers
- Die Ankunftszeit
- Die Lehrkraft, aus dessen Unterricht der Schüler gekommen ist
- Die Anzahl Vorfälle des Schülers inklusive dem ausgewählten
- Kommentare/Gründe

Am unteren Rand der Detailbox ist eine Leiste, welche sich mit fortschreitender Anwesenheit füllt. Ist diese Leiste voll, so sind 15min erreicht. Dies bedeutet, dass der Schüler abhängig von seinem Rückkehrplan zurück in den Unterricht darf. Erst dann ist der Entlassen Button ( 6 ) freigeschaltet.

### 3.2.1 Schüler eintragen

Um einen ankommenden Schüler im TRManager einzutragen, wählt man zunächst im linken Bereich ( 4 ) per Klick die gewünschte Klasse aus, woraufhin sich rechts daneben der betroffene Schüler aus der Liste aller Schüler der Klasse auswählen lässt. Die graue Hinterlegung markiert die aktuelle Auswahl. Hat man die Auswahl getroffen, ist ein Klick auf „VORFALL EINTRAGEN“ ( 5 ) der nächste Schritt.

Daraufhin öffnet sich ein Fenster namens „VORFALL ANLEGEN“:

Hier wählt man den Lehrer aus, welcher den Besuch des Schülers im Trainingsraum veranlasst hat und kann im Kommentarfeld Gründe für den Aufenthalt oder sonstige Bemerkungen hinzufügen. Außerdem steht eine Auswahlliste mit den gängigen Gründen unter dem Kommentarfeld zur Verfügung. Mit einem Klick auf „HINZUFÜGEN“ wird der Vorfall in die Tagesansicht aufgenommen.

Ist der Schüler zum *dritten* Mal im Trainingsraum, öffnet sich beim Hinzufügen des Schülers ein Fenster. Dieses weist den Lehrer darauf hin, dass ein Elterngespräch zu veranlassen ist.

Beim *fünften* Aufenthalt eines Schülers im Trainingsraum erscheint beim Eintragen des Schülers eine Meldung, dass eine Klassenkonferenz von Nöten ist.

Abbildung 4: Vorfall anlegen

### 3.2.2 Schüler entlassen

Um einen Schüler, dessen Aufenthaltszeit im Trainingsraum mindestens 15 Minuten beträgt (siehe Tagesansicht), aus dem Trainingsraum zu entlassen, muss dieser zunächst in der Tagesansicht per Klick ausgewählt werden (graue Hinterlegung markiert die Auswahl). Durch einen Klick auf „SCHÜLER ENTLASSEN“ ( 6 ) wird der entsprechende Schüler aus der Tagesansicht entfernt und ist somit bereit, den Trainingsraum zu verlassen.

*Achtung:* Das Entlassen eines Schülers vor Ablauf der 15 Minuten ist technisch nicht möglich.

### 3.2.3 Rückkehrer

Um einen Rückkehrer einzutragen klickt man im mittleren Bereich des Hauptfensters auf „RÜCKKEHRER“ ( 7 ). Daraufhin öffnet sich ein Fenster in dem die Schüler angezeigt werden, die am selben Tag bereits im Trainingsraum waren. Dort wählt man den entsprechenden Schüler aus und klickt auf „EINTRAGEN“. Er erscheint nun wie jeder andere Vorfall in der Tagesübersicht.

## 3.3 Adminfunktionen

Nach erfolgreicher Anmeldung als Admin (s. Kapitel 3.1.2) sind im Hauptfenster drei zusätzliche Reiter verfügbar:

- Administration
- Einstellungen
- Datensätze verwalten



Abbildung 5: Reiter des Hauptfenster

## Import

**Wichtig:** Bevor die neuen Schülerdaten importiert werden können, müssen die Daten des TRManagers exportiert (siehe [3.3.1: EXPORT](#)) werden. Zur Prävention von Fehlern wurde dem Import ein Export bestehender Daten vorgeschaltet, falls dieser nicht vorher manuell durchgeführt wurde.

Um den Import auszuführen, ist ein Klick auf **„DATEN IMPORTIEREN“** im Reiter „Administration“ erforderlich. Wurden die aktuellen Daten des Programms noch nicht exportiert, wird ein Dialog geöffnet, der den Nutzer dazu auffordert, die alten Daten zu sichern. Durch einen Klick auf **„OK“** wird ein Vorgang eingeleitet, der dem Export entspricht (s. Abschnitt 3.3.2). Sind die Daten exportiert, öffnet sich ein neues Fenster, in welchem die Auswahl des Verzeichnisses der entsprechenden Importdatei erfolgt. Entspricht die Importdatei dem richtigen Format, so werden nun im Reiter **„TRAININGSRAUM“** die importierten Schülerdaten (Klassen, Namen) angezeigt.

### 3.3.1 Export

Vor dem Importieren von Daten müssen zwangsläufig zunächst die Daten exportiert werden. Dazu steht aktuell kein separater Button zur Verfügung. Zu Beginn des Export öffnet sich ein Fenster, in dem das Zielverzeichnis auszuwählen ist. Nach der Auswahl des Verzeichnisses werden durch einen Klick auf **„OK“** die aktuellen Daten des Programms im Zielverzeichnis gesichert.

### 3.3.2 Einstellungen

In dem Reiter Einstellungen sind unter anderem die Verbindungseinstellungen anzugeben. Befindet sich der Server mit den Daten auf der gleichen Maschine (Standard) sind die Einstellungen wie folgt:

- Adresse: **„LOCALHOST“**
- Port: **„8080“**
- Einstiegs-Endpoint: **„TRMANAGER“**

Zusätzlich können für spätere Features die Unterrichtszeiten in Form von Schulstundenlänge, Schulbeginn und Pausenzeiten angegeben werden. Zum Hinzufügen von Pausen ist das Menü am unteren Rand des Fenster zu verwenden. Zunächst den Startzeitpunkt der Pause an Stelle von **„ZEITPUNKT“** und die Länge bei **„LÄNGE“** eintragen. Durch Klicken auf **„PAUSE HINZUFÜGEN“** wird die Pause der Liste hinzufügen.

**Abbildung 6: Einstellungen**

Wurden alle gewünschten Änderungen vorgenommen muss noch über den Button **„Konfiguration Speichern“** die Einstellungen persistent übernommen werden.

### 3.3.3 Datensätze verwalten

Die Verwaltung von Datensätzen befindet sich im gleichnamigen Reiter („DATENSÄTZE VERWALTEN“). Dieser Reiter ermöglicht unter anderem die Zeitunabhängige Manipulation von Vorfalleinträgen.

Über die Buttons „A“ bis „D“ können die unterschiedlichen Arten von Datensätze ausgewählt werden. Die aktuelle Auswahl ist an der darüber liegenden Liste erkennbar. Ist eine Variante ausgewählt kann über die Buttons „E“ bis „G“ eine entsprechende Aktion vorgenommen werden.

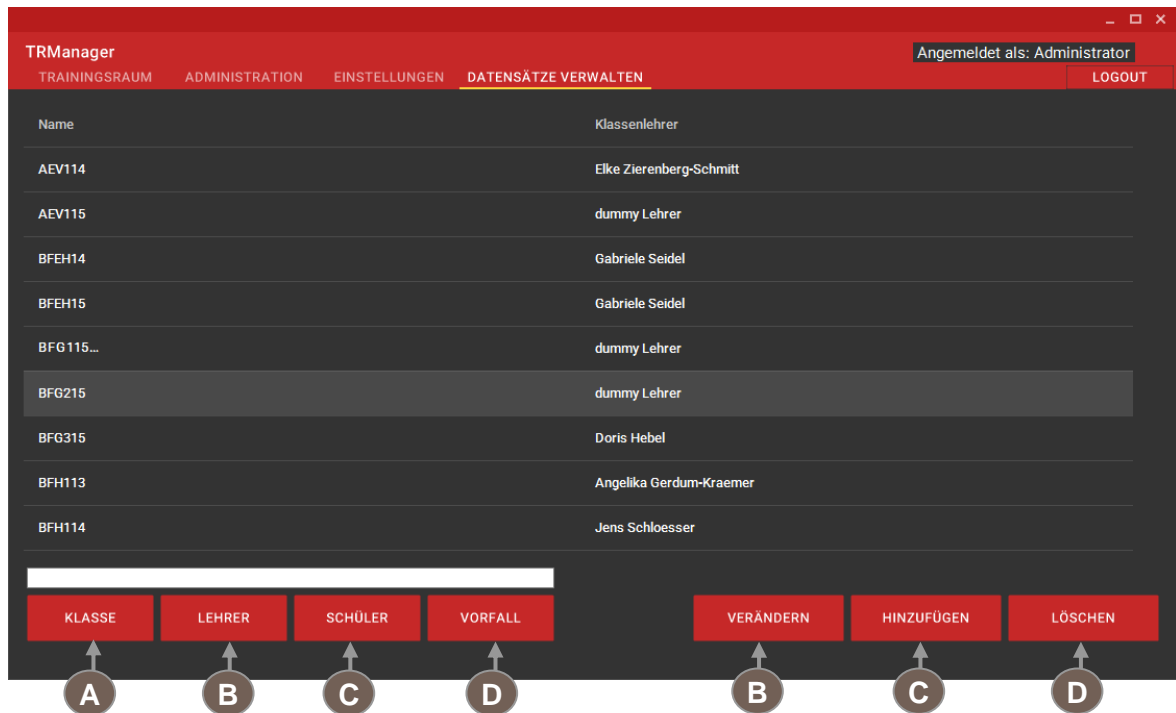


Abbildung 7: Datensätze verwalten

#### Verändern

Bevor ein Datensatz verändert werden kann, muss dieser zunächst in der Liste ausgewählt werden. Auch hier weist die graue Hinterlegung auf die aktuelle Auswahl hin. Anschließend kann der Datensatz in dem erscheinenden Menüfenster bearbeitet werden.

#### Hinzufügen

Zum Hinzufügen eines Datensatzes muss nach der Auswahl der Datenart lediglich dem Menü zum „HINZUFÜGEN“ gefolgt werden.

#### Löschen

Bevor ein Datensatz gelöscht werden kann, muss dieser zunächst in der Liste ausgewählt werden. Auch hier weist die graue Hinterlegung auf die aktuelle Auswahl hin.

Nach einer Bestätigung der Wahl zum Löschen, wird der entsprechende Datensatz aus dem Datenbestand entfernt.