

RESTful API mit Java Spring

Fallbeispiel einer API mit Spring Boot und ORM/Hibernate unter
Berücksichtigung von REST-Prinzipien

BACHELORARBEIT

MARC RAEMY

August 2020

Unter der Aufsicht von:

Prof. Dr. Jacques PASQUIER-ROCHA

and

Pascal GREMAUD

Software Engineering Group

Danksagung

Mein Dank geht an Prof. Dr. Jacques Pasquier-Rocha und Pascal Gremaud für die unkomplizierte und unterstützende Betreuung dieser Arbeit. Ebenso an alle Personen, die sonst in einer Form zum erfolgreichen Abschluss der Arbeit beigetragen und mich unterstützt haben. Andreas Ruppen sei gedankt für die Erstellung des LaTeX-Templates, das für diese Arbeit verwendet werden konnte.

Abstract

In dieser Arbeit wurde ein Application Programming Interface (API) unter Berücksichtigung von REST-Prinzipien mit dem Java Spring Framework und Spring Boot unter Anwendung von objektrelationalem Mapping mit der Java Persistence API (JPA) und Hibernate für den konkreten Anwendungsfall des Managements von Daten einer Hobby-Eishockeymannschaft programmiert. Die Arbeit beschreibt den Anwendungsfall, die Technologien und Konzepte sowie die Implementation der API.

Keywords: Java, Spring Framework, Spring Boot, ORM, Hibernate, REST API

Inhaltsverzeichnis

1. Einleitung	1
1.1. Motivation und Ziel	1
1.1.1. Eishockeyclub HC Keile	1
1.1.2. Anwendungsfälle einer möglichen HC Keile-Applikation	3
1.1.3. Übergeordnete Ziele und Abgrenzung des Themas	4
1.2. Aufbau der Arbeit	5
1.3. Notation and Konventionen	6
2. Daten, Anwendungsfälle und Mock-Client	7
2.1. Daten HC Keile	7
2.2. Anwendungsfälle	8
2.2.1. Anwendungsfall 1: Spielresultate lesen	10
2.2.2. Anwendungsfall 2: Scoringwerte lesen	10
2.2.3. Anwendungsfall 3: Spiel eintragen	11
2.2.4. Anwendungsfälle 4, 5 und 6: Spieleintrag korrigieren, neuen Spieler und neuen Gegner eintragen	13
2.2.5. Anwendungsfall 7: Anzahl Spiele pro Spieler lesen	14
2.3. Mock Graphical User Interface	15
3. Schema der Datenbank und Endpoints	20
3.1. Entity Relationship Model	20
3.1.1. Verbale Beschreibung	21
3.1.2. Tabellen	21
3.1.3. Domain Model vs. Entity Relationship Model (ERM)	23
3.2. Endpoints	23
3.2.1. Endpoints für mögliche HC Keile-Applikation	23
4. Spring Framework, ORM und REST	25
4.1. Spring Framework	25
4.1.1. Dependency Injection	26

4.1.2. Aspect Oriented Programming (AOP)	27
4.1.3. Spring Framework Kern	27
4.1.4. Spring Boot	28
4.1.5. Spring Projekte	29
4.2. Objekt/Relationales Mapping (ORM)	30
4.3. RESTful API	31
4.3.1. REST-Prinzipien	31
5. Programmierung und Dokumentation der API	33
5.1. Aufsetzen eines Spring Boot-Projekts	33
5.1.1. Spring Initializr	33
5.2. Quellcode	35
5.2.1. Entities	36
5.2.2. Repositories	40
5.2.3. Controller	42
5.2.4. Data Templates und Config	46
5.2.5. Main-Klasse	47
5.3. Datenbank	48
5.4. Dokumentation mit Swagger	49
6. Abschliessende Bemerkungen	52
6.1. Zusammenfassung	52
6.1.1. HC Keile	52
6.1.2. Anwendungsfälle der API	52
6.1.3. Datenmodell und Endpoints	53
6.1.4. Spring Framework, Spring Boot und ORM	54
6.1.5. Implementation der API	55
6.2. Probleme und Herausforderungen	56
6.2.1. Hoher Automatisierungsgrad von Spring	56
6.2.2. Konfiguration des objektrelationalen Mappings	56
6.2.3. Breites Themengebiet	57
A. Abkürzungen	58
B. Quellcode und Dokumente	60
B.1. Main-Klasse	60
B.2. Package config	63
B.3. Package datatemplates	64
B.4. Package entities	71
B.5. Package repositories	82
B.6. Package controller	84

B.7. pom.xml-File	108
B.8. ReadMe-File	109
C. Lizenz	111
Literaturverzeichnis	112
Webreferenzen	112

Abbildungsverzeichnis

1.1. Mannschaft HC Keile	2
1.2. Screenshot Beispiel Notiz Daten von Spiel HC Keile	3
2.1. Anwendungsfall-Diagramm	9
2.2. Prozess Scoringwerte lesen bisher und mit möglicher App	11
2.3. Prozess Spiel eintragen bisher	12
2.4. Prozess Spiel eintragen - mit App	12
2.5. Prozesse Spieler oder Gegner eintragen und Spieleintrag korrigieren	13
2.6. Prozess Anzahl Spiele pro Spieler zählen - bisher	15
2.7. Mock GUI - Startseite	16
2.8. Mock GUI - Spiel eintragen	17
2.9. Mock GUI - Topscorer und Resultate lesen	18
3.1. Entitäten-Beziehungsmodell	21
4.1. Spring Framework Core	28
5.1. Spring Initializr Onlinetool	34
5.2. Ordnerstruktur, Packages, Interfaces und Klassen	36
5.3. Sub- und Superinterfaces von JpaRepository, Javadoc	40
5.4. h2-console	49
5.5. Swagger Interface	51
6.1. Anwendungsfall-Diagramm	53
6.2. Spring Framework Core	54
6.3. Ordnerstruktur, Packages, Interfaces und Klassen	55

Tabellenverzeichnis

2.1. Daten HC Keile	8
3.1. Tabelle Game	22
3.2. Tabelle Goal HC Keile	22
3.3. Tabelle Player	22
3.4. Tabelle Opponent	22
3.5. Tabelle Games played	23
3.6. Endpoints für HC Keile-Applikation	24
5.1. Unterstützte Keywords in JPA für Query-Definition über den Methodennamen	42
6.1. Endpoints für HC Keile-Applikation	54

Listings

5.1. Klasse Game	36
5.2. Interface JpaRepository	40
5.3. Klasse PlayerController	42
5.4. pom.xml Jackson Dependencies	46
5.5. Klasse GoalTemplate	46
5.6. Main-Klasse KeileStatsApplication	47
5.7. Maven Dependencies Swagger	49
5.8. Swagger Configuration Class	50
6.1. snippet Endpoint GET-Request auf Tor-Entität	55
B.1. Klasse Main vollständig	60
B.2. Klasse SwaggerConfig vollständig	63
B.3. Klasse GameTemplate vollständig	64
B.4. Klasse GoalTemplate vollständig	65
B.5. Klasse OpponentTemplate vollständig	66
B.6. Klasse PlayerTemplate vollständig	67
B.7. Klasse ScoringDataTemplate vollständig	69
B.8. Klasse Game vollständig	71
B.9. Klasse Goal vollständig	74
B.10.Klasse Opponent vollständig	77
B.11.Klasse Player vollständig	78
B.12.Interface GameRepository vollständig	82
B.13.Interface GoalRepository vollständig	83
B.14.Interface OpponentRepository vollständig	83
B.15.Interface PlayerRepository vollständig	83
B.16.Klasse GameController vollständig	84
B.17.Klasse GoalController vollständig	95
B.18.Klasse OpponentController vollständig	101
B.19.Klasse PlayerController vollständig	103

B.20.pom.xml-File	108
B.21.ReadMe-File	109

1

Einleitung

1.1. Motivation und Ziel	1
1.1.1. Eishockeyclub HC Keile	1
1.1.2. Anwendungsfälle einer möglichen HC Keile-Applikation	3
1.1.3. Übergeordnete Ziele und Abgrenzung des Themas	4
1.2. Aufbau der Arbeit	5
1.3. Notation and Konventionen	6

1.1. Motivation und Ziel

Das Ziel der vorliegenden Arbeit ist es, ein Application Programming Interface (API) zu programmieren, das von einer Web-Applikation, die den Hobby-Eishockeyclub HC Keile beim Management seiner Daten unterstützt, genutzt werden könnte. In diesem Kapitel wird der HC Keile und seine Aktivitäten näher vorgestellt sowie das Ziel und der Aufbau dieser Arbeit erläutert.

1.1.1. Eishockeyclub HC Keile

In der Eishockeyszene in der Schweiz gibt es eine Vielzahl von Mannschaften, die ausserhalb von Verbands- und Meisterschaftsstrukturen regelmässig Freundschaftsspiele bestreiten. Das wird „wilde Liga“ oder „Plausch-Cup“ genannt. Der HC Keile ist eine Mannschaft aus Freiburg in der Schweiz, die in dieser Form den Eishockeysport seit 2011 als Hobby ausübt. Die Mannschaft spielt nur Matches und macht keine Trainings. Die Spiele des HC Keile finden in unregelmässigen Abständen am frühen Sonntagabend in der Eishalle in Freiburg statt.

Die meisten Spieler sind nebenbei in anderen Vereinen oder auf privater Basis sportlich aktiv. Der Kern der Mannschaft besteht aus einer Gruppe von Freunden, die aus dem freiburgischen Sensebezirk stammen und sich seit der Kindheit kennen oder sich in ihrer Zeit als aktive Fussballer bei den Fussballclubs SC Düringen und FC Freiburg kennengelernt haben.

Der HC Keile führt Scoringstatistiken zu Team und Spielern. Diese werden an der jährlichen Generalversammlung präsentiert. Der Club hat eine ausgeprägte Tradition des nicht ganz ernst gemeinten „Selbst-Kultes“, mit einer Facebook-Seite, einer Whatsapp-Gruppe, diversen Awards und Pokalen, die ebenfalls an der Generalversammlung verliehen werden. Die Statistiken dienen in erster Linie der Unterhaltung und dem Spass.



Abbildung 1.1.: Mannschaft HC Keile

Rollen und Aufgaben

An den Spielen des HC Keile werden nebst dem Resultat und Namen des Gegners die Torschützen und Assistgeber sowie die anwesenden Spieler des HC Keile notiert. Die Anzahl Spiele pro Spieler wird auch verwendet zur Verrechnung der Kosten für die Eismiete unter den Spielern.

Es gibt verschiedene Ämter im Club. Eines ist das Amt des Datenmanagers. Er notiert jeweils an den Spielen, welche Spieler des HC Keile das Spiel absolviert haben, wer die Tore geschossen hat und wer die Assists gegeben hat, sowie das Resultat und gegen welche Mannschaft gespielt wurde. Die Statistiken trägt er zu Hause von Hand in einer Excel-Tabelle ein, die als Datenbank der HC Keil-Statistiken dient.

An der Generalversammlung präsentiert diese Person jeweils Auswertungen dieser Statistiken, die er in den Excel-Tabellen vorgenommen hat. Das sind etwa die Scorerstatistiken, Siegesquoten oder Anzahl gespielte Spiele der Spieler.

Ein weiteres Amt ist jenes des Spiele-Organisators. Er organisiert die Spiele und Gegner und führt Doodle-Umfragen, in denen sich die Spieler für die Spiele eintragen können. Der Kassier der Mannschaft liest aus diesen Doodle-Umfragen, welcher Spieler wie viele Spiele

absolviert hat, um am Ende des Jahres den Spielern entsprechend der Anzahl gespielter Spiele eine Rechnung für ihren Beitrag an die Eismiete zu stellen.

1.1.2. Anwendungsfälle einer möglichen HC Keile-Applikation

Ziel der API ist das Speichern, Erheben und Kommunizieren der Daten zu erleichtern. Sie soll ermöglichen, dass die Daten künftig statt in einer Excel-Tabelle in einer Online-Datenbank gespeichert werden können, dass sie am Spiel direkt online in die Datenbank eingetragen werden können und dass sie jederzeit und jederorts konsultiert werden können. Im Folgenden werden die Anwendungsfälle (engl. „Use Cases“) einer möglichen Applikation beschrieben, welche die in dieser Arbeit programmierte API nutzen könnte.

Anwendungsfall 1: Daten der Spiele erheben

Der erste Anwendungsfall ist das Erheben der Daten an den Spielen. Die Daten werden bisher während den Spielen von Hand notiert (Abbildung 1.2. zeigt eine solche Notiz). Die Datenmanager*in trägt diese danach zu Hause in eine Excel Tabelle ein. Somit werden die Daten ein weiteres Mal von Hand notiert. Es würde Aufwand ersparen, wenn man, via eines Smartphones oder Tablets, die Daten via Internet direkt in eine Datenbank eintragen könnte und so nur einmal notieren müsste. Beispielsweise mit einem Webformulars via Browser.

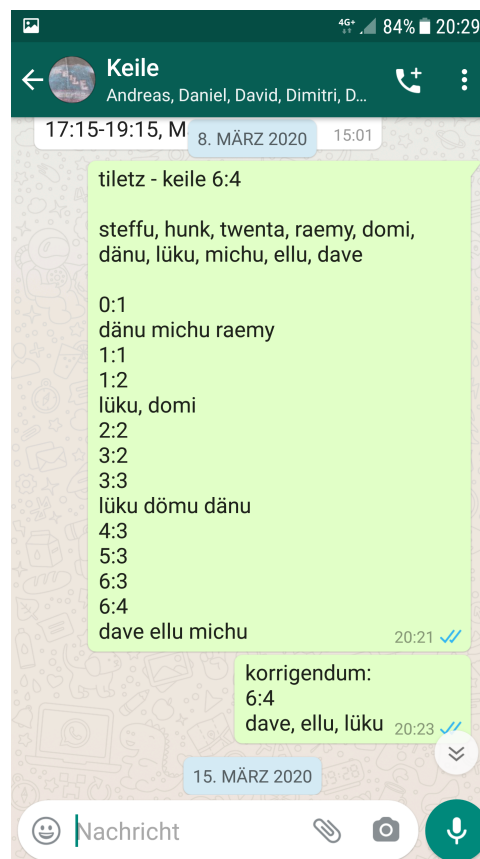


Abbildung 1.2.: Screenshot Beispiel Notiz Daten von Spiel HC Keile

Daraus ergeben sich weitere Anwendungsfälle. Erstens, wenn der Eintrag fehlerhaft war, den Spieleintrag zu korrigieren und zweitens, wenn Spieler am Spiel waren, die noch nicht in der Datenbank gespeichert sind, oder eine gegnerische Mannschaft, die noch nicht gespeichert ist, diese Entitäten in die Datenbank einzutragen.

Anwendungsfall 2: Daten der Spiele und Spieler lesen

Da es zweitens die Spieler interessiert, die Scorerliste, die Resultate und Anzahl absolvierte Spiele zu verfolgen, ist ein weiterer Anwendungsfall, die Scoringstatistiken der Spieler sowie die Resultate der Spiele darzustellen. Somit könnten die Statistiken laufend und von überall her online gelesen werden, statt nur einmal pro Jahr an der Generalversammlung.

Anwendungsfall 3: Anzahl Spiele pro Spieler*in pro Saison ablesen

Der Kassier des HC Keile verschickt, um die Kosten der Miete der Eishalle zu verrechnen, am Ende jeder Saison eine individuelle Rechnung an jede Spieler gemäss der Anzahl Spiele, die er absolviert hat. Bisher liest er die Anzahl Spiele pro Spieler aus den Doodle-Umfragen, was relativ aufwändig ist.

Der dritte Anwendungsfall der API ist deshalb, dass der Kassier die Anzahl der absolvierten Spiele pro Spieler direkt ablesen kann, ohne dass er sie aus den Doodle-Einträgen zusammenzählen muss. Das würde den Abrechnungsprozess erleichtern.

Die API beschränkt ihren Anwendungsbereich auf die Spielerstatistiken und Resultate des HC Keile. Kapitel 2 beschreibt diese Anwendungsfälle im Detail. Für weitere Bereiche der Aktivitäten des HC Keile besteht im Moment kein Bedarf für eine Applikation. Bei Bedarf kann sie künftig erweitert werden. Das ist jedoch nicht Teil dieser Arbeit.

1.1.3. Übergeordnete Ziele und Abgrenzung des Themas

Übergeordnete Ziele

Das übergeordnete Ziel der Arbeit ist, die in den Vorlesungen im Softwareengineering und in den Wirtschaftsinformatikvorlesung an der Universität Freiburg erlangten Kenntnisse anhand eines Fallbeispiels in der Praxis anzuwenden. Da in den Vorlesungen die Programmiersprache Java unterrichtet wurde, wurde ein Framework in dieser Programmiersprache für die Arbeit gewählt, um die Java Kenntnisse zu vertiefen. Speziell die Kenntnisse in Web- und Applikationsentwicklung sollen angewendet und vertieft werden.

In Absprache mit Prof. Pasquier und Pascal Gremaud wurde das Java Spring Framework gewählt. Da dort häufig ORM eingesetzt wird, und da dies eine interessante und praktische Technologie ist, wurde für die Persistenzschicht ORM mit der Java Persistence API (JPA) und dem Hibernate Framework ausgewählt. Aus praktischen Gründen wird die in-Memory-Datenbank h2 genutzt. Damit sollen aktuell in der Praxis häufig verwendeten Tools für das Erstellen von Webapplikationen angewendet und kennengelernt werden.

Im Rahmen der Umsetzung werden dazu die in den Wirtschaftsinformatikvorlesungen erworbenen Kenntnisse angewendet: Entwurf und Konzeption der relationalen Datenbank mit Erstellen eines Entitäten-Beziehungsmodells und Ableiten der entsprechenden Tabellen,

Definition der Anwendungsfälle und ihre Darstellung in einem Anwendungsfalldiagramm sowie Darstellen von Prozessen mit der Business Process Model and Notation (BPMN2). Eher in den Bereich Softwareengineering gehört das Übersetzen der Anforderungen in entsprechende Endpoints der API sowie das Implementieren dieser Endpoints gemäss REST-Prinzipien. Das Thema ORM mit JPA und Hibernate nimmt dabei relativ viel Platz ein.

Abgrenzung des Themas

Es ist nicht das Ziel der Arbeit eine Fullstack-Applikation inklusive Client zu programmieren. Der Umfang der Arbeit konzentriert sich auf das Back-End und dort auf die Entwicklung der REST-API sowie ihrer Dokumentation mit Swagger. Aspekte wie Authentifizierung und Sicherheit oder das Testen mit Spring Boot sind nicht Teil der Arbeit. Die Endpoints folgen den REST-Prinzipien, die REST-Thematik steht jedoch nicht im Zentrum. Es handelt sich um eine praktische Arbeit, welche sich auf den konkreten Anwendungsfall des HC Keile konzentriert und nicht theoretische Konzepte in voller Tiefe abhandelt.

1.2. Aufbau der Arbeit

Kapitel 1: Einleitung

In der Einleitung werden das Ziel und Thema der Arbeit, ihr Aufbau sowie Informationen zu Notation und Konventionen vorgestellt.

Kapitel 2: Daten und Anwendungsfälle

Im zweiten Kapitel werden die Anwendungsfälle der hier programmierten API beschrieben sowie ein möglicher Client der API in Form eines Mock-Client skizziert.

Kapitel 3: Schema der Datenbank und Endpoints

Im dritten Kapitel wird ein Entitäten-Beziehungs-Modell sowie die daraus abgeleiteten Tabellen beschrieben. Zweitens werden daraus die für die API benötigten Endpoints abgeleitet.

Kapitel 4: Spring Framework, ORM und REST

Im vierten Kapitel werden die verwendeten Technologien und Konzepte beschrieben: Das Java Spring Framework, Spring Boot, ORM, Hibernate und REST.

Kapitel 5: Programmierung und Dokumentation der API

Im fünften Kapitel werden der Quellcode, das Aufsetzen des Projekts und die Dokumentierung der API mit Swagger vorgestellt.

Kapitel 6: Abschliessende Bemerkungen

Im abschliessenden Kapitel werden die Resultate zusammengefasst und einige Probleme und Herausforderungen der Arbeit beschrieben.

Anhang

Der Anhang enthält Literatur- und Onlinereferenzen, die Liste der Abkürzungen, eine Lizenzangabe für die Software sowie den vollständigen Quellcode.

GitHub Repository

Der Quellcode sowie alle Dokumente sind auf GitHub verfügbar unter <https://github.com/MarcRaemy/keilestats-app.git>.

1.3. Notation and Konventionen

Das Format der Arbeit basiert auf einem LaTeX-Template, das Andreas Ruppen für die Softwareengineering-Forschungsgruppe der Universität Freiburg (CH) erstellt hat. Die Notationen und Konventionen wurden grösstenteils daraus übernommen.

- Formatierung:
 - Abkürzungen werden wie folgt verwendet: Hypertext Transfer Protocol (HTTP) bei der ersten Verwendung und HTTP bei jeder weiteren Verwendung;
 - Webadressen in folgender Form: `http://localhost:8080/api`;
 - Format Quellcode:

```
1 public double division(int _x, int _y) {  
2     double result;  
3     result = _x / _y;  
4     return result;  
5 }
```
- Die Arbeit ist in sechs Kapitel unterteilt, die jeweils Unterkapitel enthalten. Jedes Unterkapitel hat Paragraphen, welche eine gedankliche Einheit repräsentieren.
- Grafiken, Tabellen und Auflistungen sind innerhalb eines Kapitels nummeriert. Beispielsweise wird eine Referenz auf Grafik j des Kapitels i nummeriert als *Abbildung i.j.*
- Bezüglich der geschlechtlichen Form wird, da nur in seltenen Fällen Frauen oder intersexuelle Menschen beim HC Keile mitspielen, der Kürze und Einfachheit halber die männliche Form verwendet. Frauen und intersexuelle Personen sind dabei mitgemeint. An gewissen Stellen wird in der Arbeit auch die Konvention mit weiblicher Form und Stern verwendet, z.B. „Entwickler*innen“, welche Männer und intersexuelle Personen einschliesst.

2

Daten, Anwendungsfälle und Mock-Client

2.1. Daten HC Keile	7
2.2. Anwendungsfälle	8
2.2.1. Anwendungsfall 1: Spielresultate lesen	10
2.2.2. Anwendungsfall 2: Scoringwerte lesen	10
2.2.3. Anwendungsfall 3: Spiel eintragen	11
2.2.4. Anwendungsfälle 4, 5 und 6: Spieleintrag korrigieren, neuen Spieler und neuen Gegner eintragen	13
2.2.5. Anwendungsfall 7: Anzahl Spiele pro Spieler lesen	14
2.3. Mock Graphical User Interface	15

In diesem Kapitel werden zuerst die Daten des HC Keile kurz beschrieben. Der zweite Abschnitt beschreibt die Anwendungsfälle („Use Cases“) im Detail. Im dritten Abschnitt wird eine graphische Benutzeroberfläche (GUI) eines möglichen Client skizziert, um die Anwendungsfälle zusätzlich zu illustrieren.

2.1. Daten HC Keile

Wie bereits erwähnt, notiert der HC Keile bei jedem Spiel die anwesenden Spieler, wer die gegnerische Mannschaft war, das Resultat sowie die Torschützen und Assistgeber der Tore des HC Keile (nicht des Gegners). Tabelle 2.1. zeigt einen Auszug aus den Rohdaten. Es ist ein CSV-Auszug aus einer Excel-Datei, in der die Daten des HC Keile zur Zeit lokal gespeichert sind.

ID	MatSaison	MatchNr	SpAka	SpPos	SpT	SpA1	SpA2	GeT	SpRes
1145	201617	22	Stefu	G	0	0	0	4	gewonnen
1146	201617	22	Twenta	V	0	0	1	0	gewonnen
1147	201617	22	Doemu	V	0	1	0	0	gewonnen
1148	201617	22	Reamy	V	0	0	0	0	gewonnen
1149	201617	22	Hunk	V	0	0	0	0	gewonnen
1150	201617	22	Elu	S	2	0	1	0	gewonnen
1151	201617	22	Michu	S	1	2	0	0	gewonnen
1152	201617	22	Chraebli	S	2	0	1	0	gewonnen
1153	201617	22	Dave	C	2	0	0	0	gewonnen
1154	201617	22	Sebi	S	0	0	0	0	gewonnen
1155	201617	22	Flogge	C	0	1	0	0	gewonnen
1156	201617	22	Pavel	S	2	0	1	0	gewonnen
1157	201617	22	Roman	S	1	1	0	0	gewonnen
1158	201718	1	StefuG	G	0	0	0	8	verloren
1159	201718	1	Stephu	V	0	0	0	0	verloren
1160	201718	1	Twenta	V	0	0	0	0	verloren
1161	201718	1	Michu	S	0	2	0	0	verloren
1162	201718	1	Fongs	V	0	0	0	0	verloren
1163	201718	1	Sebi	G	2	0	0	0	verloren
1164	201718	1	Flogge	G	0	0	1	0	verloren
1165	201718	1	Dave	C	0	0	1	0	verloren
1166	201718	1	Katja	S	0	0	0	0	verloren
...									

Tabelle 2.1.: Daten HC Keile

ID = ID des Eintrags/Excel-Zeile, MatSaison = Saison, MatchNr = Nummer Match der laufenden Saison (1 = erstes Spiel der Saison), SpAka = Rufname des Spielers, SpPos = Position (G = Goalie, S = Sturm (Flügel), C = Center, V = Verteidiger), SpT = erzielte Tore, SpA1 = Anzahl erste Assists, SpA2 = Anzahl zweite Assists, GeT = Gegentore (nur für Goalie), SpRes = Resultat -> gewonnen, verloren oder unentschieden

Tabelle 2.1. zeigt exemplarisch einen Auszug aus den Daten. Es sind die Daten des letzten Spiels der Saison 2016/2017 und des ersten Spiels der Saison 2017/2018. Der gesamte Datensatz umfasst rund 140 Spiele aus neun Saisons. Seit dem ersten Spiel im Jahr 2011 sind alle Daten erfasst.

2.2. Anwendungsfälle

Das Anwendungsfall-Diagramm in Abbildung 2.1. gibt einen Überblick über die Anwendungsfälle.

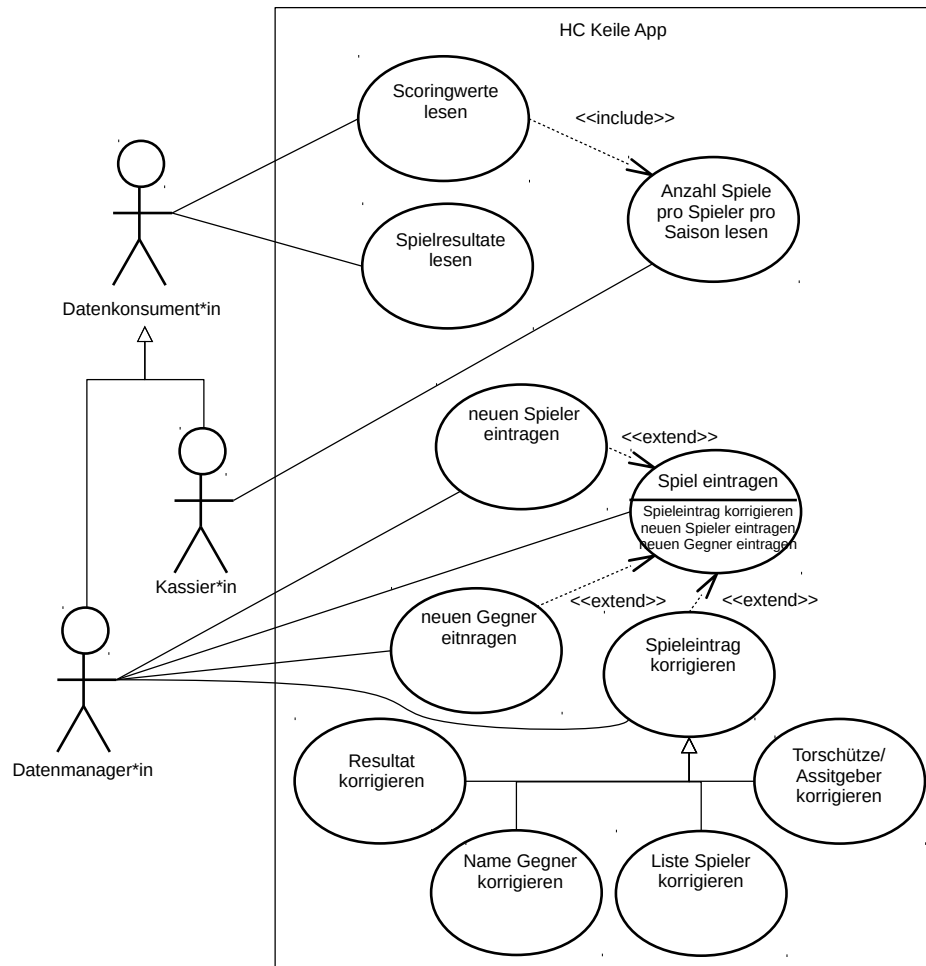


Abbildung 2.1.: Anwendungsfall-Diagramm

HC Keile-App

Das Rechteck repräsentiert die Applikation, das System. Alles, was in der Applikation passiert, ist innerhalb des Rechtecks dargestellt. Wie erwähnt ist in diesem Fall das System eine mögliche

künftige Applikation für den HC Keile, welche das hier programmierte Application Programming Interface (API) nutzen würde.

Akteure

Die primären Akteure sind die Spieler des HC Keile. Sie sind im Diagramm als „Datenkonsument*in“ bezeichnet. Ausserhalb des HC Keile dürfte es wenig Personen geben, die ein Interesse an den Spielresultaten und Scoringzahlen des HC Keile haben.

Unter den Datenkonsumenten gibt es Personen mit Spezialfunktionen, welche das System in besonderer Weise nutzen. Das ist erstens der Kassier, der aus den Daten die Anzahl Spiele pro Spieler pro Saison abliest und anhand dessen die Kosten für die Eismiete pro Spieler berechnet. Zweitens ist es der Datenmanager, welcher die Scoringdaten, Resultate, Namen der Gegner und wer an den Spielen anwesend war, an den Spiele erhebt und für den HC Keile sammelt, speichert und präsentiert. Sie sind im Diagramm in einer Generalisierungsbeziehung zur Akteurin „Datenkonsument*in“ dargestellt.

Assoziationen

Zwischen den Akteur*innen und den Use Cases bestehen Beziehungen, auch Assoziationen genannt. Die Akteur*in Datenkonsument*in liest Matchresultate und Scoringstatistiken, sie ist deshalb mit diesen Anwendungsfällen assoziiert.

Die Kassier*in ist mit dem Anwendungsfall „Anzahl Spiele pro Spieler pro Saison lesen“ assoziiert. Dieser Anwendungsfall steht in einer „include“-Beziehung zum Anwendungsfall „Scoringdaten lesen“. In den Scoringdaten sind auch die Anzahl Spiele, die eine Spieler*in in der Saison gespielt hat, enthalten. Deshalb schliesst der Anwendungsfall „Scoringdaten lesen“ den Anwendungsfall „Anzahl Spiele pro Spieler pro Saison lesen“ ein.

Die Datenmanager*in ist mit den Anwendungsfällen „Spiel eintragen“, „neuen Spieler eintragen“, „neuen Gegner eintragen“ und „Spieleintrag korrigieren“ assoziiert. Beim Eintrag eines Spiels kann es sein, dass ein neuer Gegner oder ein neuer Spieler in die Datenbank eingetragen werden muss, oder dass ein Eintrag im Nachhinein korrigiert werden muss (siehe z.B. Korrektur in Abbildung 1.2. in Kapitel 1). Dies ist jedoch nur manchmal der Fall, in vielen Fällen nicht. Das ist die Bedeutung der „extend“-Beziehungen mit umgekehrtem Pfeil beim Anwendungsfall „Spiel eintragen“.

Beim Anwendungsfall „Spieleintrag korrigieren“ gibt es vier Einzelfälle: Entweder muss die Spielerliste, das Resultat, der Name des Gegners oder der Namen eines Torschützen oder Assistsgebers korrigiert werden. Diese Anwendungsfälle stehen deshalb in einer Generalisierungsbeziehung zum Anwendungsfall „Spieleintrag korrigieren“.

2.2.1. Anwendungsfall 1: Spielresultate lesen

Die Spieler des HC Keile wollen die Resultate der Spiele konsultieren können. Dies ist der erste Use Case „Spielresultate lesen“. Die Datenkonsument*in konsultiert im System die Spielresultate des HC Keile. Das System stellt dabei das Datum, Gegner sowie das Resultat der Spiele zur Verfügung.

2.2.2. Anwendungsfall 2: Scoringwerte lesen

Zweitens möchten die Spieler*innen des HC Keile wissen, wer am meisten Tore und Assists erzielt hat und wie viele er oder sie selber erzielt hat. Der zweite Use Case ist also das Konsultieren der

Scoringwerte. Von Interesse sind dabei die Anzahl Tore, die jede Spieler*in geschossen hat, die Anzahl erster und zweiter Assists und die Summe davon als Gesamtzahl Scorerpunkte.

Zusätzlich ist relevant, wie viele Spiele die Person gespielt hat. Diese Daten könnten von einem Client beispielsweise in einer Tabelle dargestellt werden, ähnlich dem Statistikportal der Schweizer Eishockeymeisterschaft (welches jedoch viel mehr Daten umfasst) <https://www.sihf.ch/de/game-center/national-league/#/mashup/players/player/points/desc/page/0/2019>.

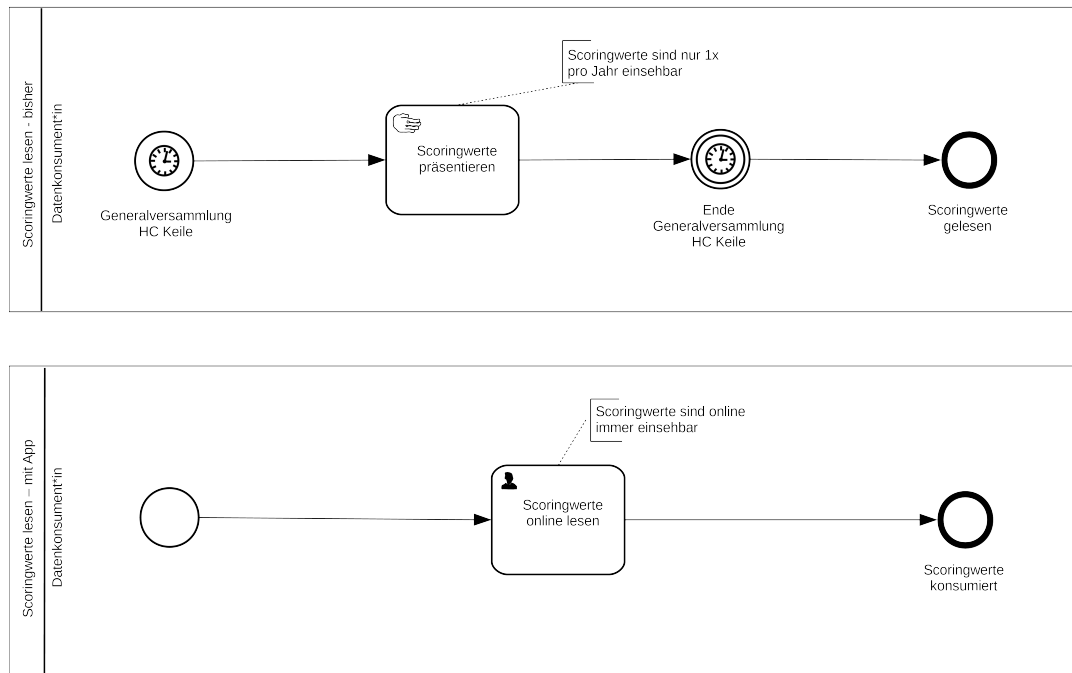


Abbildung 2.2.: Prozess Scoringwerte lesen bisher und mit möglicher App

Abbildung 2.2. zeigt den Prozess, die Scoringwerte zu lesen, wie er im Moment ist: die Scoringwerte werden einmal pro Jahr an der Generalversammlung präsentiert und sind sonst nicht einsehbar, und wie er mit einer möglichen App sein könnte: die Scoringwerte sind online von überall und jederzeit einsehbar.

2.2.3. Anwendungsfall 3: Spiel eintragen

Bei den Spielen des HC Keile werden die Statistiken bisher jeweils von Hand notiert. Dies ist in Abbildung 2.3. durch das Hand-Symbol oben Links in den Tasks signalisiert. Folgende Daten werden wie erwähnt erhoben: Name des Gegners, Anzahl Tore des Gegners, Anzahl Tore HC Keile, Spieler des HC Keile, die am Spiel teilgenommen haben sowie Namen der Torschützen und Assistenten des HC Keile. Über die Namen der Torschützen des Gegners wird nicht Buch geführt.

Die Daten werden, wie erwähnt, bisher dann noch einmal von Hand von der Datenmanager*in in eine Excel-Datei eingetragen. Die erste Notiz dient nur der Übermittlung. Die Excel-Datei dient als Datenspeicher und ist nur lokal auf einem privaten Rechner abgespeichert und nicht online zugänglich.

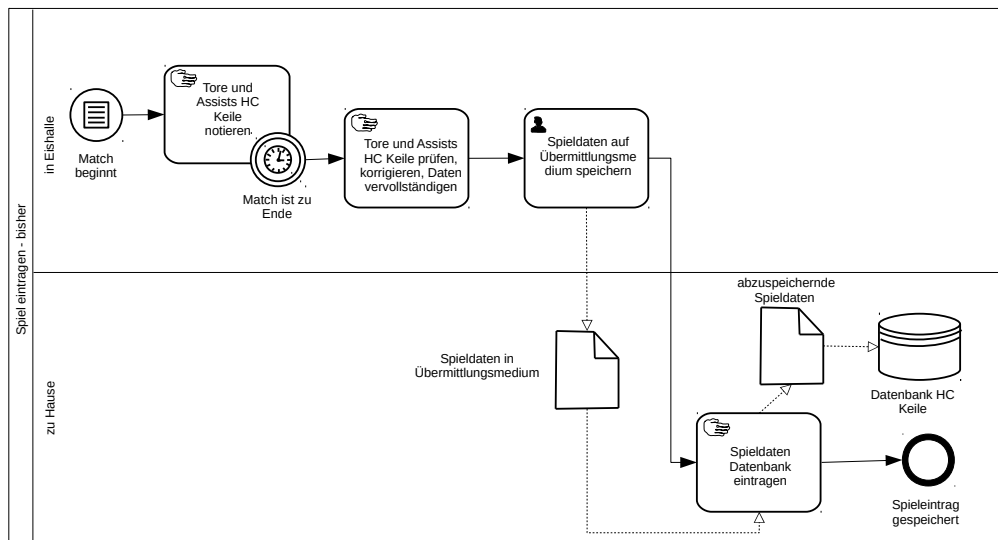


Abbildung 2.3.: Prozess Spiel eintragen bisher

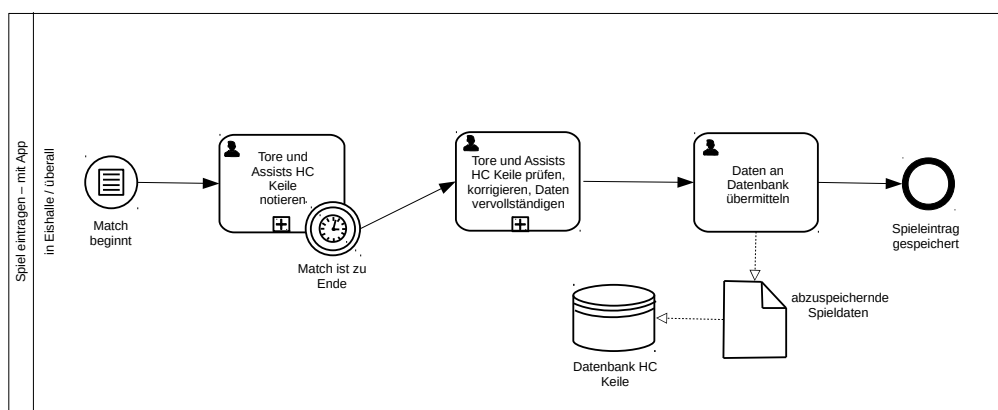


Abbildung 2.4.: Prozess Spiel eintragen - mit App

Abbildung 2.3. stellt den bisherigen Prozess dar. Abbildung 2.4. zeigt, wie der Prozess mit einer möglichen Applikation unterstützt werden könnte. Die Datenmanager*in gibt die Daten nach dem

Spiel via eines Web-Formulars, das die hier programmierte API nutzt, online in eine Datenbank ein. Somit müssen die Daten nur einmal aufgeschrieben werden. Zudem könnte die Applikation auch als Notiz-Applikation für das Notieren der Daten während des Spiels gebraucht werden.

2.2.4. Anwendungsfälle 4, 5 und 6: Spieleintrag korrigieren, neuen Spieler und neuen Gegner eintragen

Nun kann es sein, dass ein Spiel eingetragen wird, bei dem ein Eintrag fehlerhaft ist. Zum Beispiel könnte bei einem Tor versehentlich der falsche Spieler als Assistgeber vermerkt sein, oder bei den teilnehmenden Spielern ist ein falscher oder fehlender Name auf der Liste. Daher ist ein weiterer Anwendungsfall „Spieleintrag korrigieren“. Er erweitert den Anwendungsfall „Spiel eintragen“, da er in manchen Fällen, aber nicht immer, wenn ein Spiel eingetragen wird, eintritt.

Der Prozess, die Daten der Spiele zu notieren, beginnt während dem Spiel. Fällt ein Tor, werden die Torschützen und Assistgeber notiert. Nach dem Spiel notiert er das Resultat, Namen des Gegners, welche Spieler des HC Keile den Match gespielt haben sowie überprüft und vervollständigt die Torschütz*innen und Assistgeber*innen. Es wird nicht immer bei allen Toren während des Spiels gesehen, wer beteiligt war. Dies wird nach dem Spiel nachgefragt und ergänzt. Abbildung 2.5. stellt diesen Prozess graphisch dar. Die folgenden Abschnitte beschreiben die einzelnen Elemente.

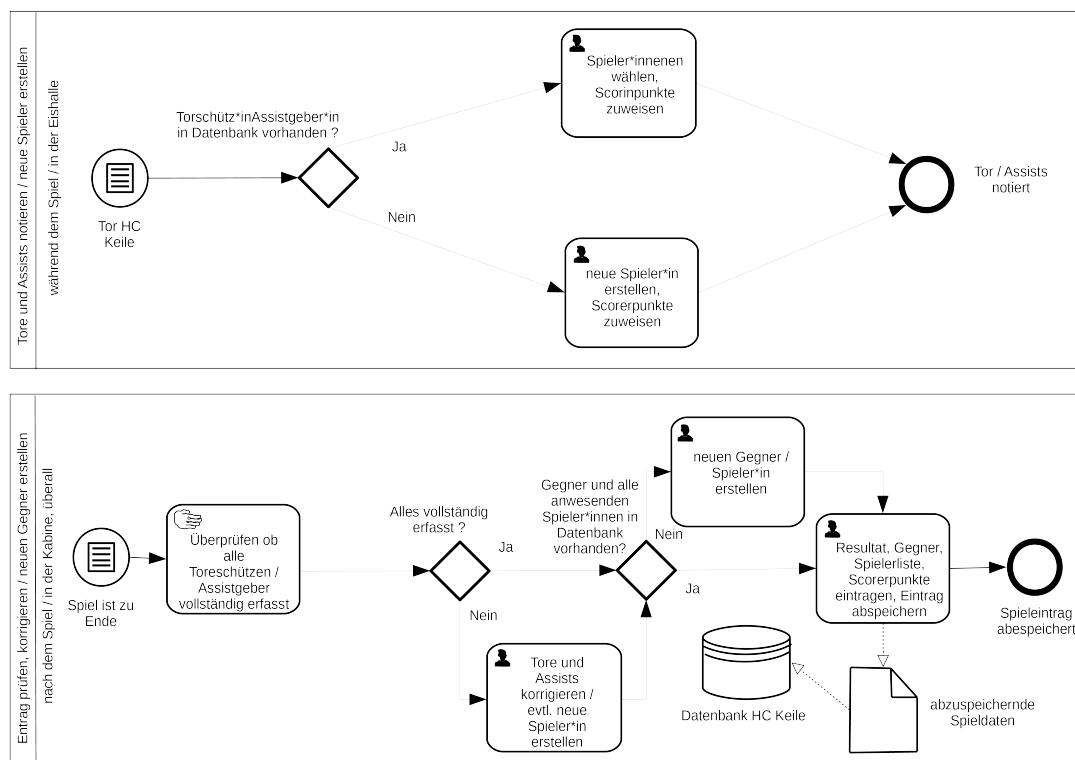


Abbildung 2.5.: Prozesse Spieler oder Gegner eintragen und Spieleintrag korrigieren

Neuen Spieler eintragen

Angenommen, die Notiz würde mit einer HC Keile-Applikation auf dem Smartphone vorgenommen. Vorausgesetzt es besteht eine Internetverbindung, könnten die am Tor beteiligten Spieler*innen aus der Datenbank geladen und ausgewählt werden. Wenn jedoch das Tor von einer Spieler*in

erzielt würde, welche noch nicht in der Datenbank eingetragen ist, muss diese Spieler*in neu erstellt werden. Dies sollte möglichst schnell und einfach gehen, da dies während dem Spiel, zwischen den Einsätzen erledigt wird. Dazu sollte die Spieler*in nur mit Vornamen und Namen erstellt werden können, ohne, dass weitere Daten wie Telefonnummer oder Adresse, die man eintragen könnte, eingetragen werden müssen. Gleichzeitig sollte es möglich sein, die Daten später zu ergänzen, oder eine Spieler*in von Anfang an mit den vollständigen Daten zu erstellen.

Neuen Gegner eintragen

Nach dem Spiel wird der Spieleintrag vervollständigt. Dazu wird nebst dem Resultat, den Tor-schütz*innen und Assistgeber*innen und der Liste der Spieler*innen, die am Spiel teilgenommen haben auch der Name des Gegners notiert. Im Datenmodell ist der Gegner als eigene Entität modelliert. Wenn das Spiel gegen einen Gegner stattfand, der bereits in der Datenbank eingetragen ist, könnte dieser aus der Datenbank geladen und ausgewählt werden. Andernfalls muss ein neuer Gegner erstellt werden.

Spieleintrag korrigieren

Es ist möglich, dass nach dem Spiel (entweder noch in der Kabine, oder ein paar Stunden später oder an einem späteren Tag) bei einem Eintrag zu einem Spiel etwas ändern muss. Sei das, dass man bei einem Tor einen anderen Namen als Assistgeber*in angeben muss, das Resultat falsch eingetippt wurde, bei der Liste der teilnehmenden Spieler eine Änderung gemacht werden muss, oder sogar, dass die falsche gegnerische Mannschaft angegeben wurde. Dies bedeutet, dass der Spieleintrag geladen und Änderungen an den entsprechenden Attributen gemacht werden können sollten.

2.2.5. Anwendungsfall 7: Anzahl Spiele pro Spieler lesen

Für die Spiele des HC Keile muss die Eishalle gemietet werden. Die Kosten dafür belaufen sich auf rund 600 Franken pro Spiel und werden unter den beiden Mannschaften aufgeteilt. Beim HC Keile zahlt jede Spieler*in ihren Anteil an den Spielen, an denen sie teilgenommen hat.

Das Geld wird nicht direkt bar an den Spielen eingezogen, sondern am Ende der Saison mit einer individuellen Rechnung für jede Spieler*in von der Kassier*in eingezogen. Dafür muss die Kassier*in für jede Spieler*in wissen, wie viele Spiele sie in der abgelaufenen Saison absolviert hat. Abbildung 2.6. zeigt den Prozess des Zählens der Spiele, wie er bisher durchgeführt wird. Die Anzahl Spiele wird manuell aus den Doodle-Umfragen gelesen und zusammengezählt. Die Abbildung soll illustrieren, dass der Ablauf so relativ aufwändig ist.

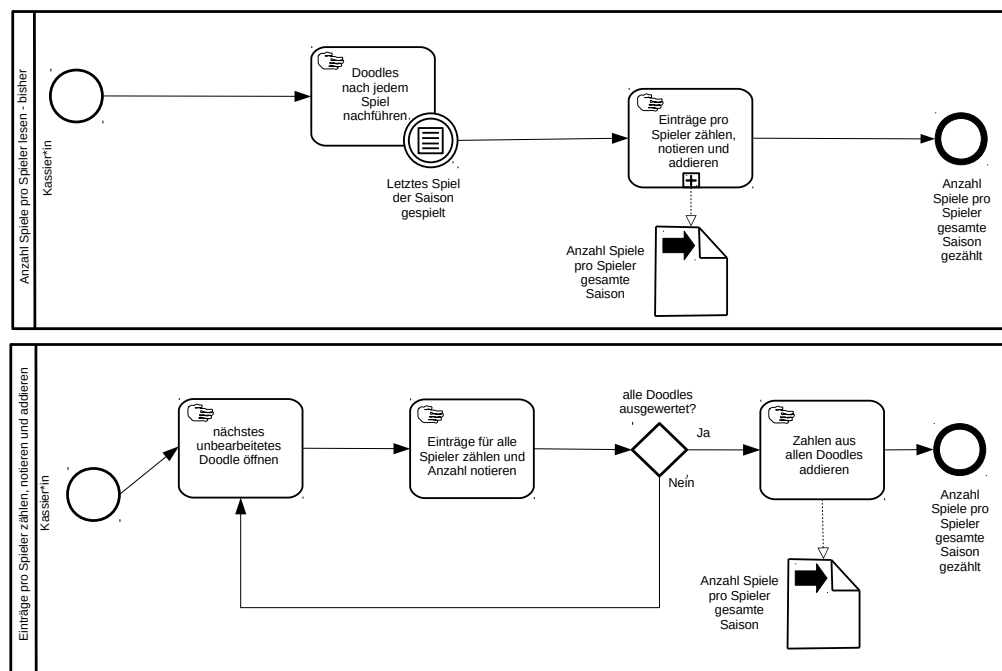


Abbildung 2.6.: Prozess Anzahl Spiele pro Spieler zählen - bisher

Bisher hat die Person dafür in den Doodles nachgezählt, wer an wie vielen Spielen in der Saison teilgenommen hatte. Es gibt rund vier Doodles pro Saison, in denen sich die Spieler für die nächsten ca. fünf Spiele einschreiben müssen. Es musste jedes Doodle jeweils von Hand ausgezählt werden, was doch relativ umständlich ist. Mit einer möglichen HC Keile-Applikation könnte der Kassier künftig direkt online die Spielerstatistiken öffnen und auf einen Blick sehen, welche Spieler wie viele Spiele absolviert haben (dies ist hier nicht in einem Prozessdiagramm dargestellt).

2.3. Mock Graphical User Interface

Dieses Kapitel zeigt, wie ein möglicher Client aussehen könnte, mit dem diese Anwendungsfälle umgesetzt werden könnten. Abbildung 2.7. zeigt die Startseite der Applikation. Abbildung 2.8. zeigt, wie die Seite „Spiel eintragen“ und damit das Formular, um die Daten an einem Spiel einzutragen, aussehen könnte. Abbildung 2.9. zeigt, wie die Anwendungsfälle „Scoringwerte lesen“, „Spielresultate lesen“ und „Spieleintrag korrigieren“ umgesetzt werden könnten.

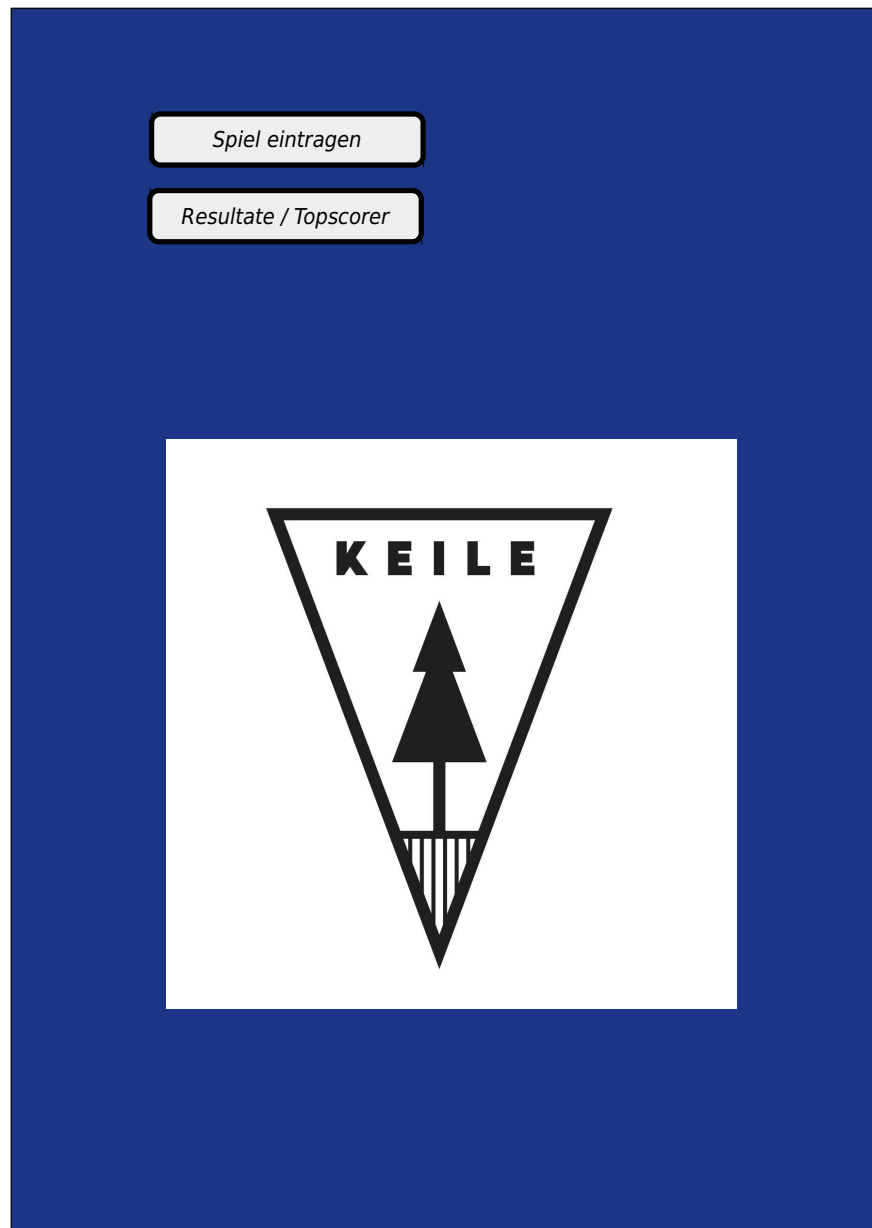


Abbildung 2.7.: Mock GUI - Startseite

Auf der Startseite könnte man beispielsweise wählen, ob man die Statistiken und Resultate lesen, oder ob man ein Spiel eintragen will.

GAMEDAY...!

HC KEILE

vs.

Neuen Gegner eintragen...

Datum

Tore HC Keile

Tore Gegner

Spieler HC Keile

Neuen Spieler eintragen..

Tor #1 HC Keile

Tor #2 HC Keile

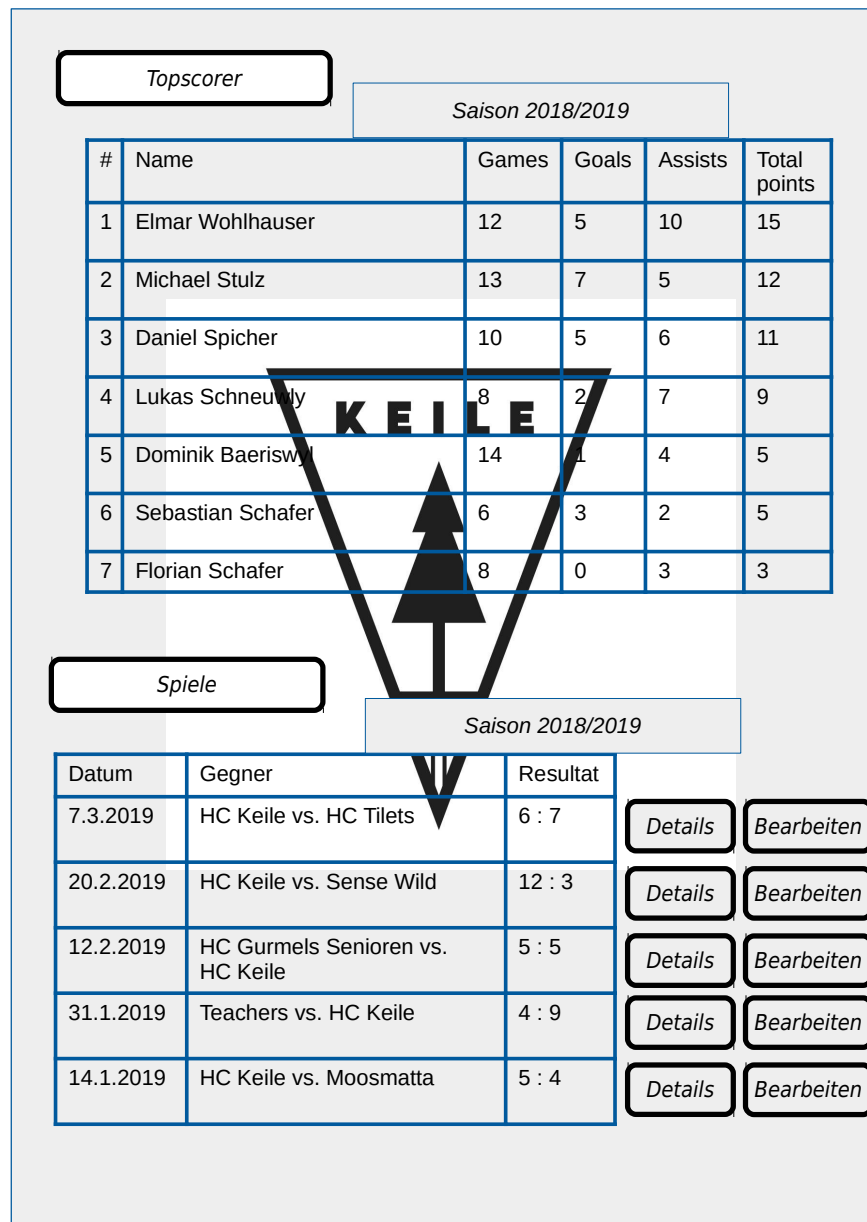
+ Tor Keile...

Spiel speichern...

Abbildung 2.8.: Mock GUI - Spiel eintragen

Auf der Seite „Spiel eintragen“ wird zuoberst beispielsweise der Gegner eingetragen. Das könnte so gestaltet sein, dass, wenn man in das Feld klickt, die Gegner, die in der Datenbank gespeichert sind, zur Auswahl angezeigt werden und man dann den entsprechenden Gegner auswählen könnte. Spielt man zum ersten Mal gegen den Gegner, kann man ihn neu erstellen mit dem Button

darunter. Ähnliches gilt für die Einträge der Tore und der Spieler*innen. Mit dem Button „Spiel speichern...“ wird das Spiel in die Datenbank übermittelt.



Topscorer

Saison 2018/2019

#	Name	Games	Goals	Assists	Total points
1	Elmar Wohlhauser	12	5	10	15
2	Michael Stulz	13	7	5	12
3	Daniel Spicher	10	5	6	11
4	Lukas Schneuwly	8	2	7	9
5	Dominik Baeriswyl	14	1	4	5
6	Sebastian Schafer	6	3	2	5
7	Florian Schafer	8	0	3	3

Spiele

Saison 2018/2019

Datum	Gegner	Resultat
7.3.2019	HC Keile vs. HC Tilets	6 : 7
20.2.2019	HC Keile vs. Sense Wild	12 : 3
12.2.2019	HC Gurmels Senioren vs. HC Keile	5 : 5
31.1.2019	Teachers vs. HC Keile	4 : 9
14.1.2019	HC Keile vs. Moosmatta	5 : 4

Details

Details

Details

Details

Details

Bearbeiten

Bearbeiten

Bearbeiten

Bearbeiten

Bearbeiten

Abbildung 2.9.: Mock GUI - Topscorer und Resultate lesen

Für die Anzeige der Daten könnte beispielsweise in der oberen Hälfte die Scoringwerte nach Saison geordnet angezeigt werden, inklusive Anzahl Spiele pro Spieler*in in dieser Saison. In der

unteren Hälfte könnten die Spiele angezeigt werden. Dort könnten mit dem „Bearbeiten“-Button Spieleinträge korrigiert und mit dem „Details“-Button die Torschütz*innen und Assitgeber*innen des Spiels sowie die anwesenden Spieler*innen angezeigt werden.

3

Schema der Datenbank und Endpoints

3.1. Entity Relationship Model	20
3.1.1. Verbale Beschreibung	21
3.1.2. Tabellen	21
3.1.3. Domain Model vs. Entity Relationship Model (ERM)	23
3.2. Endpoints	23
3.2.1. Endpoints für mögliche HC Keile-Applikation	23

In diesem Kapitel wird ein Entitäten-Beziehungsmodell für die API entworfen und die Endpoints aus diesem sowie aus den Anwendungsfällen aus Kapitel 2 abgeleitet. Zuerst wird das Modell dargestellt, beschrieben und die Tabellen für eine relationale Datenbank daraus abgeleitet. Dann werden die Endpoints der API definiert und beschrieben.

3.1. Entity Relationship Model

In einem Entitäten-Beziehungsmodell (ERM) wird der Anwendungsbereich der Applikation respektive der Datenbank modelliert. Es gibt verschiedene Möglichkeiten, wie man die Modellierung umsetzen kann. Abbildung 3.1. zeigt eine Möglichkeit für das vorliegende Beispiel. Die rechteckigen Felder sind Entitäten, die Rauten bezeichnen die Beziehungen zwischen den Entitäten. Die Zahlen bei den Beziehungen geben an, ob die Beziehung eines (1), eines oder keines (c) , mehrere (m) oder keines oder mehrere (mc) Elemente umfasst.

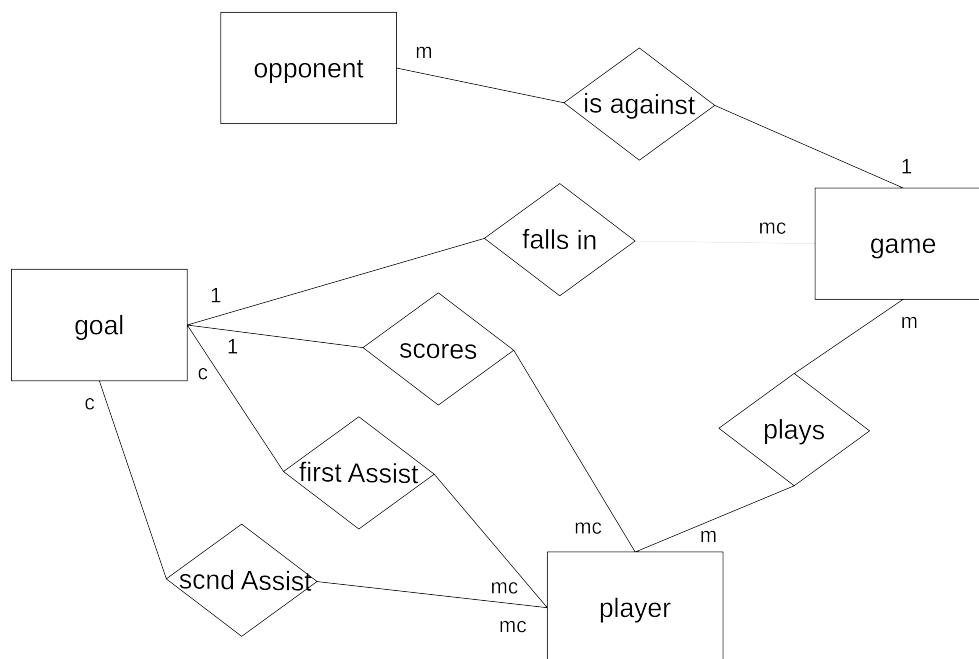


Abbildung 3.1.: Entitäten-Beziehungsmodell

3.1.1. Verbale Beschreibung

Mehrere Spieler*innen (player) spielen ein Spiel (game). In einem Spiel werden keine, eines oder mehrere Tore des HC Keile (goal) erzielt. Einem Spiel ist genau eine gegnerische Mannschaft (opponent) zugeordnet. Diese kann dies in einem oder mehreren Spielen sein.

Eine Spieler*in spielt eines oder mehrere Spiele. Sie kann keine, eines oder mehrere Tore, erste (first assist) oder zweite (second assist) Assists erzielen. Ein (erstes oder zweites) Assists gehört genau zu einem Tor des HC Keile und wird genau von einer Spieler*in erzielt. Ein Tor hat keines oder ein erstes Assists und keines oder ein zweites Assists und wird von genau einer Spieler*in erzielt. Das Resultat des Spiels und damit die Tore der gegnerischen Mannschaft sind als Attribut der Entität Spiel modelliert.

Zwischen den Entitäten "player" und "game" besteht die einzige komplex-komplexe Beziehung des Modells. Mehrere Spieler*innen spielen ein Spiel und eine Spieler*in spielt (meistens) mehrere Spiele. Diese Relation wird deshalb in einer eigenen Tabelle abgebildet.

3.1.2. Tabellen

Daraus kann man für die Modellierung der Datenbank folgende Tabellen ableiten: Je eine Tabelle pro Entität: "game", "goal", "opponent" und "player", dazu eine Tabelle für die komplex-komplexe Beziehung "player plays game". Die restlichen Relationen werden mittels Fremdschlüssel in den assoziierten Tabellen modelliert. Die Tabellen 3.1. bis 3.5. zeigen die abgeleiteten Tabellen mit ihren Attributen.

Game

attribute	format	primary key	foreign key	not null
game_id	INT	✓		✓
game_date	CHAR			✓
nb_goals_opponent	INT			✓
nb_goals_keile	INT			
opponent_id	INT		✓	

Tabelle 3.1.: Tabelle Game

Goal

attribute	format	primary key	foreign key	not null
goal_id	INT	✓		✓
game_id	INT		✓	✓
scorer_id	INT		✓	✓
first_assist_id	INT		✓	
second_assist_id	INT		✓	

Tabelle 3.2.: Tabelle Goal HC Keile

Player

attribute	format	primary key	foreign key	not null
player_id	INT	✓		✓
firstname	CHAR			
lastname	CHAR			
address	CHAR			
phone	CHAR			
email	CHAR			
position	CHAR			

Tabelle 3.3.: Tabelle Player

Opponent

attribute	format	primary key	foreign key	not null
opponent_id	INT			✓
name	CHAR			

Tabelle 3.4.: Tabelle Opponent

Games played				
attribute	format	primary key	foreign key	not null
player_id	INT	✓	✓	✓
game_id	INT	✓	✓	✓

Tabelle 3.5.: Tabelle Games played

3.1.3. Domain Model vs. ERM

Die Literatur zu Hibernate [1, S. 64] beschreibt als Ausgangspunkt für die Softwareentwicklung mit Hibernate das Erstellen eines Domain Models. Im Buch von Bauer und King [1, S. 64] wird das Domain Model als abstrakte, formalisierte Repräsentation der realen Entitäten, welche die Software abbilden und unterstützen soll, sowie deren logischen Beziehungen untereinander dargestellt. Daraus werden in der objektorientierten Programmierung die Klassen und Unterklassen sowie ihre Verbindungen im Programm abgeleitet.

Das Entitäten-Beziehungsmodell hat eine ähnliche Herangehensweise und Funktion. Es stellt ebenfalls die realen Entitäten und ihre logischen Beziehungen dar, welche formalisiert und abstrahiert dargestellt werden. Es wird jedoch verwendet, um daraus die Struktur der relationalen Datenbank abzuleiten und zu normalisieren (siehe Kapitel 3). In einem Domain Model können noch mehr Arten von Verbindungen und Funktionalitäten dargestellt werden, als im Entitäten-Beziehungsmodell [53].

Für diese Arbeit bestehen nicht riesige Unterschiede zwischen einem Domain Model und einem Entitäten-Beziehungsmodell. Bauer und King erwähnen, dass Hibernate nicht nur vom Domain Model aus, sondern auch vom Datenbankmodell ausgehend eingesetzt werden kann [1, S. 64]. Bei grösseren Applikationen sei das Ausgehen vom Domain Model die bessere Wahl, da Hibernate helfe, Objekte in Relationen zu übersetzen und nicht umgekehrt [1, S. 64]. Hier wird dennoch als Ausgangspunkt für die Programmierung das Entitäten-Beziehungsmodell wie in Kapitel 3 dargestellt, verwendet, da, wie erwähnt, die Unterschiede hier sehr klein sind.

3.2. Endpoints

Ein Endpoint einer API ist die URI, über die eine bestimmte Ressource aufgerufen oder abgespeichert wird [52], verbunden mit einer entsprechenden HTTP-Standardmethode. Im World Wide Web sind dies die HTTP-Methoden (z.B. GET, PUT, POST, DELETE). Zwei HTTP-Requests können dieselbe URI haben, aber verschiedene Methoden, z.B. GET und POST.

Um einen Endpoint zu definieren, muss man die Methode (z.B. GET) spezifizieren, sowie die URI, auf die die Abfrage angewendet wird. Weiter ist zu definieren, welche Daten in welchem Format übergeben werden. Für die hier erstellte API werden alle Daten im Multipurpose Internet Mail Extensions (MIME)-Format „application/json“ versendet und verarbeitet.

3.2.1. Endpoints für mögliche HC Keile-Applikation

Basierend auf den im Kapitel 2 beschriebenen Anwendungsfällen und den Prinzipien der API-Gestaltung nach REST werden im nächsten Abschnitt die in dieser Arbeit erstellten Endpoints für die API dargestellt. Auf die REST-Prinzipien wird in Kapitel 4.3. näher eingegangen.

Tilkov et al. 2015 [7] weisen in Ihrem Buch „HTTP und REST“ [7] darauf hin, dass für die Gestaltung gemäss REST-Prinzipien die Ressourcen, welche mit der API bearbeitet und zur Verfügung gestellt werden, gemäss den Anforderungen und Zielen der Anwendung und nicht zwingend starr deckungsgleich mit den Entitäten der Datenbank gestaltet werden sollen (auch wenn sich das oft überschneidet), die Ressourcen sind getrennt von ihrer Implementation auf Serverseite zu betrachten. Tabelle 3.6. leitet aus der vorhergehenden Beschreibung der Anwendungsfälle die Ressourcen, URI's und Methoden, welche die Enpoints konstituieren, ab.

Ressource	URI	Methode	Anwendungsfälle
Spieleintrag, Liste Spiele	\games	POST, GET	Spiel eintragen, Spielresultate lesen
Einzelnes Spiel	\games\{game_id}	GET, PUT, DELETE	Spieleintrag korrigieren, Spieldetails lesen
Liste Spieler an Spiel	\games\{game_id}\players	PUT	Liste Spieler korrigieren
Liste Tore	\goals	POST, GET	Tor/e eintragen
Einzelnes Tor an Spiel	\goals\{goal_id}	GET, PUT, DELETE	Torschütz*in / Assistgeber*in korrigieren
Spieler*innen	\players	GET, POST	neue Spieler*in eintragen
Einzelne Spieler*in	\players\{player_id}	GET, PUT, DELETE	Eintrag Spieler*in korrigieren, bearbeiten
Tore pro Spieler*in	\players\{player_id}\goals	GET	Scoringwerte lesen, Tore
Assists pro Spieler*in	\players\{player_id}\assists	GET	Scoringwerte lesen, Assists
Spiele pro Spieler*in	\players\{player_id}\games	GET	Scoringwerte lesen, Anzahl Spiele pro Spieler*in lesen
Topscoerliste	\players\scoringtable	GET	Scoringwerte lesen, Übersicht
Gegner	\opponents	POST, GET	neuen Gegner eintragen, Liste Gegner anzeigen
Einzelner Gegner	\opponents\{opponent_id}	PUT, GET, DELETE	Eintrag Gegner korrigieren, bearbeiten

Tabelle 3.6.: Endpoints für HC Keile-Applikation

Eine Grundregel der Ressourcengestaltung nach REST ist, dass auch Subressourcen (z.B. Liste von Spieler*innen, die an einem Spiel teilgenommen haben als Subressource eines Spiels) als eigene Ressourcen abgebildet werden sollen, sofern sie für Anwendungsfälle relevant sind. [7]. Ein weiteres Prinzip - generell der Softwareentwicklung - besagt, dass man zuerst nur das implementieren soll, was man sicher braucht, und nicht auf Vorrat für mögliche Funktionalitäten programmieren, die dann evtl. doch nicht verwendet werden. Daher fokussieren die hier programmierten Endpoints hauptsächlich auf die in Kapitel 2 beschriebenen Anwendungsfälle.

4

Spring Framework, ORM und REST

4.1. Spring Framework	25
4.1.1. Dependency Injection	26
4.1.2. Aspect Oriented Programming (AOP)	27
4.1.3. Spring Framework Kern	27
4.1.4. Spring Boot	28
4.1.5. Spring Projekte	29
4.2. Objekt/Relationales Mapping (ORM)	30
4.3. RESTful API	31
4.3.1. REST-Prinzipien	31

In diesem Kapitel werden zuerst das Spring Framework und Spring Boot vorgestellt, dann wird auf das Thema objektrelationales Mapping eingegangen und schliesslich werden die REST-Prinzipien dargestellt.

4.1. Spring Framework

Das Java open source Framework Spring bietet eine Implementation formalisierter best practices der Softwarearchitektur im Business-Kontext an [4]. Die erste Version von Spring wurde im Jahr 2002 vom australischen Programmierer und Musikwissenschaftler Rod Johnson in seiner Arbeit „Expert One-on-One J2EE Design and Development“ [3] entwickelt. Johnson versucht darin eine Antwort zu finden auf die Komplexität der Entwicklung von Unternehmens-Anwendungen, an die anspruchsvolle Anforderungen gestellt werden, wie komplexe, datenintensive Datenbanktransaktionen, Sicherheit und Authentifizierung, Zugang übers Web oder Monitoring der Performance und der Sicherheit [8].

Er suchte nach einem Weg, solche Businessapplikationen auf möglichst einfache und schnelle Weise programmieren zu können und diese flexibel, leicht wart- und modifizierbar und einfach erweiterbar zu halten [8]. Daraus entstand das Spring-Framework. Spring erreicht diese Ziele zu einem wichtigen Teil, indem es die Konzepte der Dependency Injection (DI) und des Aspect Oriented Programming AOP umsetzt [4].

Vorläufer: JavaBeans

In den Neunzigerjahren wurde von Sun Microsystems die JavaBeans-Spezifikation entwickelt [54]. Dabei handelt es sich um ein Softwarekomponentenmodell, das Entwicklungsrichtlinien enthält, welche die Wiederverwendung von einfachen Java-Objekten und ihre Zusammenstellung zu komplexeren Applikationen vereinfachen sollte [8, S.4]. Sie entstanden aus der Notwendigkeit heraus, Java-Klassen möglichst einfach zu instantiieren und über Remote Method Invocation (RMI) in verteilten Applikationen verwenden zu können. Die Haupteigenschaften von JavaBeans sind ein öffentlicher, parameterloser Konstruktor, Serialisierbarkeit und öffentliche Zugriffsmethoden (Getters und Setters) [54]. Dies ermöglicht ein Management der Dependencies und Instantiierungen der Applikationskomponenten durch das Framework.

Dependencies

Als Dependencies werden erstens Libraries, Dokumente, Teile anderer Programme oder Teile derselben Applikation bezeichnet, die ein Programm braucht, um ausgeführt werden zu können. Built tools helfen, die Links zu den nötigen Ressourcen herzustellen und diese verfügbar zu machen und aktuell zu halten.

Von Enterprise JavaBeans zu Spring

JavaBeans wurden ursprünglich für die Entwicklung von Benutzeroberflächen-Widgets entwickelt. Für Businessapplikationen, die aufgrund von z.B. Sicherheits-, Persistenz- oder Monitoringfunktionalitäten höhere Anforderungen stellen, war es nicht üblich, mit dieser Art von Programmierung zu arbeiten [8].

Ein erster Versuch JavaBeans im Businesskontext einzusetzen war die Enterprise Java Beans (EJB)-Spezifikation [8]. Damit sollten komplexere Anwendungen für die Bedürfnisse von Unternehmen mit Einsatz der flexiblen und leichtgewichtigen JavaBeans erstellt werden können. Die Kombination dieser Konzepte in EJB war jedoch zu praxisfern, weshalb die Entwicklung schwerfällig blieb und sich die Entwickler davon abwendeten. Grund dafür war vor allem, dass die Klassen, welche die Businesslogik implementierten, zu sehr mit externem JavaBeans-Code „verunreinigt“ wurden [8, S.4].

Parallel dazu wurde das Spring Framework entwickelt, mit dem gleichen Ziel, JavaBeans für Businessapplikationen nutzbar zu machen [8]. Der Trend der Java Entwicklung ging mit Spring weg von den komplexen EJB-Objekten hin zu simplen Plain Old Java Objects (POJO)'s und dazu, diese auch für anspruchsvollere Funktionalitäten zu verwenden. Dies jedoch ohne, dass sie davon wissen, respektive ohne, dass der Code dieser Klassen mit Code von diesen Querschnittsfunktionalitäten wie Persistenz oder Sicherheit verunreinigt wird (Separation of Concerns, Aspect Oriented Programming). Dies war vorher noch nicht möglich.

Spring ist das bekannteste und am weitesten entwickelte Framework für die Entwicklung von Businessapplikationen mit JavaBeans [8, S.4]. Der Begriff „Bean“ oder „JavaBean“ wird dabei nicht vollkommen strikt nach Spezifikation verwendet. Er bezeichnet oft einfach eine Anwendungskomponente, eine Instanz einer Klasse, die der JavaBeans-Spezifikation [2] zu grossen Teilen, aber nicht in jedem Fall zu 100% zu folgt [8, S.5].

4.1.1. Dependency Injection

Eine der wichtigsten Funktionen des Spring Frameworks ist es, loose Kopplung, statt „tight coupling“, zwischen den Objekten zu erreichen. In einer grösseren Applikation wird eine grosse

Anzahl von Klassen instantiiert. Diese Objekte hängen von anderen Objekten ab, die sie als Felder oder für Ihre Methoden benötigen, ihre „dependencies“. Das Instanzieren von Objekten und deren Übergabe an eine andere Klasse, das im Spring-Umfeld „wireing“ genannt wird, wird dabei nicht hartcodiert im Quellcode durchgeführt, sondern durch das Framework via Konfigurationsinformationen zur Laufzeit. Die Konfiguration teilt dem Framework mit, welche JavaBeans instantiiert und mit welchen anderen Objekten verknüpft werden müssen [35]. Somit sind sie lose miteinander gekoppelt und nicht „tight“. Damit kann die Applikation besser gewartet, erweitert, modifiziert und migriert werden [46].

Container Interface: Application Context

Das Ganze wird durch einen Container gemanagt. Er liest die Konfigurations-Metadaten und „injiziert“ die Instanzen und ihre Dependencies in die Beans, die er erstellt. Das Programm wird dadurch konfigurierbar, ohne den Code zu verändern. Der Container in Spring ist eine Implementation eines Interfaces mit dem Name „Application Context“ [25]. Dies ist ein zentrales Interface des Spring Frameworks.

Die Metadaten können dabei in XML, als Java Annotationen oder als Java Code selber definiert werden [45]. Eine sogenannte Bean Factory, die dem Factory Design-Pattern folgt, erstellt die Instanzen und übergibt sie an die sie aufrufenden Klassen [45].

4.1.2. AOP

Nebst der DI ist AOP ein zweites Kernfeature von Spring [8]. Aspects sind nichts anderes als spezifische Java Klassen, die Anforderungen, welche über verschiedene Schichten und Klassen der eigentlichen Applikation benötigt werden (sogenannte „cross cutting concerns“), enkapsulieren und implementieren. Dies sind typischerweise Funktionalitäten wie Sicherheit, Authentifizierung, Transaktionen oder Metrics.

Aspect Oriented Programming verfolgt das Prinzip, diese losgelöst von der Funktionalität der anderen Programmteile, d.h. der Business Logik, zu implementieren. Einzelne Klassen kümmern sich nur um ihre Kernfunktion und enthalten keinen Code von anderen Aufgaben, wie z.B. Datenbanktransaktions- oder Sicherheitsfunktionalitäten. Sie werden flexibel mit den Klassen der Business-Logik verschaltet. Dadurch verhindert man, dass in jeder Klasse einer Applikation sich wiederholender Code von nicht zu der Klasse gehörender Funktionalität enthalten ist. Dies würde zu schwer wartbaren und viel Boilerplate-Code (der gleiche Code, der in vielen Klassen immer wieder geschrieben werden muss) enthaltende Klassen und Applikationen führen [8, S.5].

4.1.3. Spring Framework Kern

Der Spring Framework-Kern ist die Basis von allem und stellt den Container zur Verfügung, in dem die Applikation gebildet wird und der für die Erstellung der Beans und die Dependency Injection zuständig ist [9, S. 26]. Dazu enthält er weitere essentielle Komponenten wie das Webframework Spring MVC, welches das Hauptframework von Spring für Webapplikationen ist, Support für eine Vielzahl von Persistenzfunktionalitäten und eine Testumgebung [9, S.26]. Abbildung 4.1. gibt eine Übersicht über die Elemente des Spring Framework-Kerns.

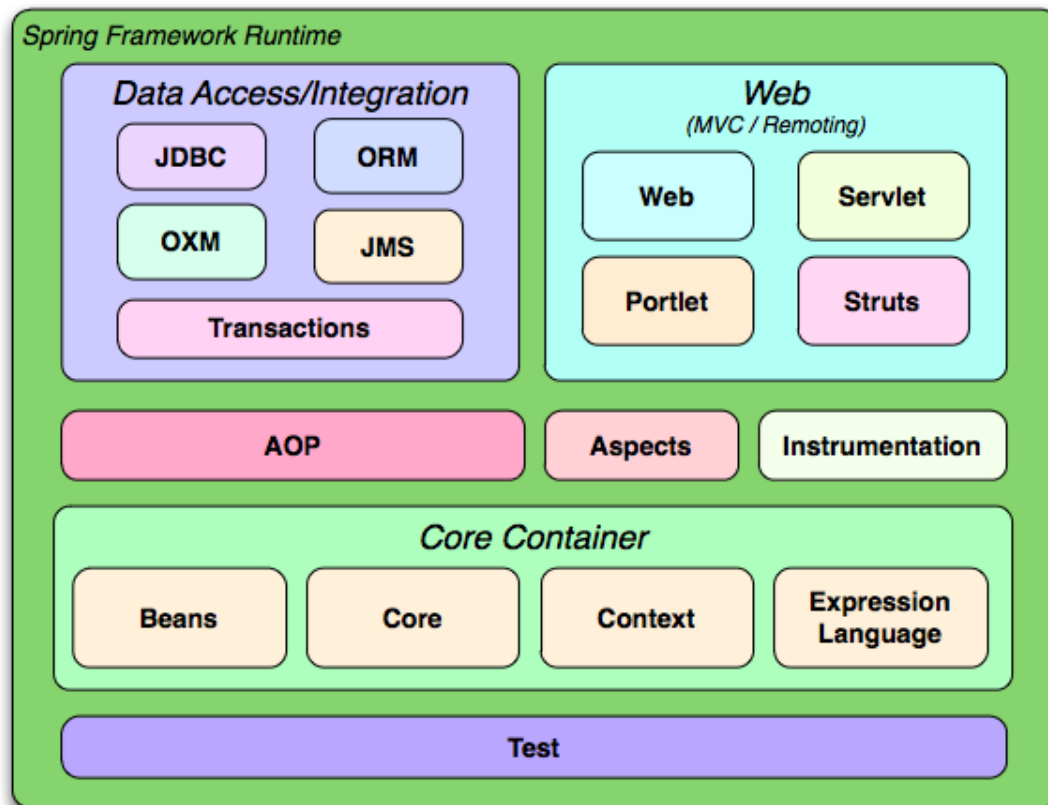


Abbildung 4.1.: Spring Framework Core

4.1.4. Spring Boot

Spring Boot ist mittlerweile essenzieller Bestandteil der Entwicklung mit Spring [9]. Es ermöglicht eine sehr schnelle und einfache Entwicklung von Applikationen, indem es eine Autokonfiguration der Applikationen zur Verfügung stellt [9], die aber leicht an spezifische Bedürfnisse angepasst werden kann.

Während das Spring-Ökosystem eine breite Palette von Frameworks für Business-Applikationen zur Verfügung stellt, dient Spring Boot dazu, mit möglichst wenig Programmier- und Konfigurationsaufwand und in möglichst kurzer Zeit eine komplette, einsatzfähige Applikation, insbesondere von Microservices, erstellen zu können [35][18]. Dies wird erreicht, indem Spring Boot eine Konfiguration der Applikation vornimmt, die best practices und Konventionen, die sich in der Industrie etabliert haben, folgt. Es setzt damit den Grundsatz „Convention over Configuration“ um [51].

Autokonfiguration

Die Autokonfiguration der Applikation ist die Hauptfunktion von Spring Boot. Es liest die Klassen und JAR-files im classpath und erstellt basierend darauf die Konfiguration [23]. Wenn beispielsweise Hibernate im classpath ist, sowie eine spezifische Datenbank, konfiguriert Spring Boot Hibernate automatisch für diese spezifische Datenbank.

Nebst der Autokonfiguration sind die Überwachung der Applikation (Metrics und health checks), das Testing und der eingebettete Tomcat-Server wesentliche Elemente einer Spring Boot-Applikation. Die Library für Metrics heisst Spring Boot Actuator. Damit kann man beispielsweise

prüfen, ob ein Service erreichbar ist, wie oft er aufgerufen wurde oder wie oft ein Aufruf fehlgeschlagen ist. Weiter stellt Spring Boot zusätzliche Unterstützung für das Testen der Applikation zur Verfügung im Vergleich zum Kern des Spring Frameworks. Dazu ermöglicht Spring Boot das flexible Spezifizieren von Umgebungseigenschaften der Applikation [9].

Spring Boot generiert dabei keinen Code und ist kein Applikations- oder Webserver [35]. Es soll es der Programmier*in lediglich ermöglichen, sich auf die Logik und die Funktionalität der Applikation zu konzentrieren, und möglichst wenig Code für das Framework oder für die Konfiguration schreiben zu müssen [9, S.26].

Integrierter Tomcat Server

Spring Boot versucht, alles, was man für eine vollständig funktionierende Applikation braucht, mitzubringen [9, S. 22]. So ist ein integrierter Tomcat Server ein Teil einer Spring Boot Applikation und wird automatisch für diese konfiguriert. Das Konzept des eingebetteten Servers ist, dass der Server im JAR-File der Applikation mitliefert wird. Er läuft somit in einem Container. Damit erspart man sich das Installieren und Verbinden der Applikation mit einem externen Server. Für Microservices, die über ein Netzwerk verknüpft sind, bietet das eine grosse Vereinfachung [35].

4.1.5. Spring Projekte

Auf Basis des Spring Framework Kerns existiert eine grosse Anzahl Projekte, die für eine Vielfalt von Funktionalitäten und Anwendungsfällen Libraries und Frameworks zur Verfügung stellen. Beispielsweise für Messaging, Unterstützung für alle geläufige Arten von Datenbanken, Webapplikationen, Sicherheitsfunktionalitäten, Cloudapplikationen, Androidapplikationen und vieles mehr. Die Liste ist beliebig erweiterbar. Einen Überblick erhält man auf der Webseite des Spring-Frameworks unter „<https://spring.io/projects/spring-framework>“.

Diese Frameworks und Libraries können mit dem Spring Initializr (siehe Kapitel 5) online sehr einfach durch Anwählen von Checkboxes je nach Bedarf heruntergeladen werden. Somit erhält man ein vorkonfiguriertes Applikationsgerüst, mit dem man in kurzer Zeit und mit wenig Code die lauffähige, gewünschte Applikation erstellen kann.

Spring ist in diesem Sinne nicht nur ein Framework, sondern ein Framework of Frameworks [8]. Im Folgenden sind einige Spring Projekte exemplarisch dargestellt. Es handelt sich aber in keiner Weise um eine vollständige Auflistung.

Spring Data

Über die Basisunterstützung für Datenbankkonnektivität des Spring Framework-Kerns hinaus bieten die Datenbank-Module von Spring Data Unterstützung für die Verbindung mit nahezu allen gängigen Datenbanktechnologien. Folgende Datenbankmodule stehen beispielsweise zur Verfügung [9][S. 75]:

- Spring Data JPA: Verbindung mit relationalen Datenbanken.
- Spring Data MongoDB: Verbindung mit einer Mongo document database.
- Spring Data Neo4j: Verbindung mit einer Neo4j Graph-Datenbank.
- Spring Data Redis: Verbindung zu einem Redis key-value store.
- Spring Data Cassandra: Verbindung zu einer Cassandra Datenbank.

Dazu bietet Spring eine einfache Art, diese Datenbankkonnektivität zu implementieren, indem einzig ein simples Java-Interface definiert werden muss [9] (siehe Kapitel 5). Spring Data Java

Persistence API (JPA) liess sich unter anderem auch von Hibernate inspirieren. JPA mit Spring ist eine der verbreitetsten Methoden zur Persistenzimplementation mit Java [8, S. 148].

Spring Security

Das Spring Security Framework ist ein umfassendes Framework, um Sicherheitsfunktionalitäten zu implementieren. Damit können beispielsweise Authentifizierung, Autorisierung oder API-Sicherungsfunktionen umgesetzt werden [9].

Spring Integration und Spring Batch

Um Applikationen mit anderen Applikationen oder mit Teilen derselben Applikation zu verbinden, bestehen bestimmte Design Patterns, die sich in der Industrie durchgesetzt haben. Spring Batch und Spring Integration bieten Implementationen dieser Patterns, mit denen Spring-basierte Applikationen diese Funktion umsetzen können [9].

Spring Cloud

Spring Cloud bietet Libraries und Frameworks, welche für die Erstellung von Microservice-basierten, cloudbasierten Applikationen benötigt werden. Eine Architektur, die mehr und mehr zum Standard wird [9]. Spring Cloud ist sehr umfangreich und deckt ein breites Feld von Anforderungen in diesem Bereich ab [9].

4.2. ORM

Im Folgenden wird erklärt, was unter ORM verstanden wird, und welche Probleme ORM löst.

Klassen in Java entsprechen nicht eins zu eins den Tabellen in relationalen Datenbanken. Dieses Problem wird auch „object/relational mismatch“ genannt. Es gibt mehrere Arten von nicht-Übereinstimmung zwischen den beiden Konzepten. ORM ist eine Lösung für dieses Problem, die aus der Praxis über Jahre hinweg entwickelt wurde [1].

Mit ORM können mittels Konfigurationsinformationen auf den Klassen der objektorientierten Programmiersprache automatisch entsprechende Tabellen für relationale Datenbanken sowie entsprechende Queries für die Operationen auf der Datenbank generiert werden. Bei früheren Datenbank-Mapping-Tools oder Libraries wie Java Database Connectivity (JDBC) musste man viel Code schreiben, die Queries aufwändig von Hand formulieren und die Weiterbearbeitung der Ergebnisse ebenfalls auf selber programmieren, um diese Funktionalität zu implementieren.

Ein weiteres Problem, wenn die Queries von Hand formuliert werden müssen, ist, dass, wenn etwas in der Datenbank verändert wird, auch alle Queries angepasst werden müssen. Das kann eine sehr aufwändige Aufgabe sein, wenn man viele und grosse Queries in einer grossen Applikation mit vielen Datenbanktabellen verwalten muss [23]. Mit Hibernate wird dieser Teil des Programmierens automatisiert. Hibernate ist eines der ersten und bekanntesten ORM-Frameworks. Es implementiert die Java Persistence API (JPA) und inspirierte die Entwicklung von JPA, bietet aber noch weitere Funktionalitäten [23].

JPA definiert Sets von Annotationen und Interfaces, mit denen das Mapping der Objekte auf die Datenbank und die Datenbankoperationen umgesetzt werden können [35]. Das Interface `JpaRepository` stellt beispielsweise eine von mehreren Abstraktionen sowie Standardimplementationen für die Basis-Datenbankoperationen (CRUD-Operationen: Create, Retrieve/Read, Update,

Delete/Destroy), zur Verfügung [23]. Es wird auch in dieser Arbeit verwendet (siehe Kapitel 5, Abschnitt 2).

Transparent Persistence

Hibernate implementiert das Konzept der „transparent persistence“. Das heisst, dass die Klasse, die in der Datenbank gespeichert wird, nichts davon wissen muss, dass sie das wird. Sie wird mit keinem Code für Datenbank-Abspeicherung verunreinigt und muss keine Interfaces implementieren oder Klassen erweitern. Das Konzept der „separation of concerns“, d.h. dass sich die Business-Logik der Applikation nicht um Dinge wie die Persistenz kümmern muss und umgekehrt, wird damit realisiert. Die Klasse wird in den Meta-Informationen der Applikation als persistente Entität deklariert.

4.3. RESTful API

Ein weiteres Konzept, das in dieser Arbeit verwendet wird, ist Representational State Transfer (REST). REST ist ein durch ein Set von Richtlinien und Prinzipien definierter Architekturstil für Schnittstellen (Interfaces) zwischen verteilten Systemen und Applikationen [7]. Er wurde von Roy Fielding, der in der 90er Jahren bereits an der Standardisierung des HTTP-Protokolls mitgearbeitet hatte, in seiner im Jahr 2000 beendeten Dissertation beschrieben und erarbeitet [7]. REST beschreibt eine Abstraktion der Konzepte und Prinzipien, die HTTP zu Grunde liegen. HTTP und darauf aufbauend das World Wide Web sind also eine konkrete und wohl die bekannteste Implementation einer REST-Architektur [7]. Die allgemeinen Ziele für verteilte Systeme, die REST zu erreichen hilft, sind lose Kopplung, Interoperabilität, Wiederverwendung, Performance und Skalierbarkeit [7].

Im Vergleich zum ebenfalls häufig bei Webservices verwendeten Standard Simple Object Access Protocol (SOAP) sind REST-implementierende Abfragen oft schneller und datenärmer, da ein weniger grosser Overhead mitgeschickt wird als bei SOAP, welches auf das relativ datenreiche XML-Format festgesetzt ist. REST API's sind also häufig ein leichtgewichtiger Ansatz für Webservices, der zunehmend verbreiteter wird [5]. REST und SOAP lassen sich aber gar nicht direkt miteinander Vergleichen, da REST einen Architekturstil bezeichnet, SOAP aber eine konkrete, XML-basierte Schnittstellen-Standardisierung [7].

4.3.1. REST-Prinzipien

Auf ein Minimum reduziert, lässt sich REST in fünf Kernprinzipien zusammenfassen [7]:

- Ressourcen mit eindeutiger Identifikation:

Mittels Unified Resource Identifier (URI) werden im Web alle Ressourcen (statische wie dynamische) eindeutig identifiziert. Dieses Prinzip wird auch bei RESTful-Webservices angewendet, indem beispielsweise ein Kunde oder eine Bestellung mit einer URI eindeutig identifiziert wird. Dadurch kann ein Link darauf versendet werden, was für viele Anwendungsfälle nützlich ist. Die Ressourcen sollten dabei so bezeichnet werden, dass sie für Menschen gut lesbar sind. Jede für den Anwendungsfall nützliche Ressource kann dabei mit einer URI identifiziert und somit zugänglich und weiter verarbeitbar gemacht werden [7].

- Verknüpfung/Hypermedia:

Ein zweites Prinzip des REST-Architekturstils ist die Verwendung von Verknüpfungen von Ressourcen um Applikationen zu steuern und zu gestalten und um Ressourcen zu verbinden, wie im obigen Abschnitt mit dem Beispiel des Links auf den neu erstellten Kunden skizziert.

Ein Ausdruck, der in diesem Zusammenhang verwendet wird, ist „Hypermedia as the Engine of Application State (HATEOAS)“. Mittels des Zusendens unterschiedlicher Links an den Client in Abhängigkeit der Funktion und des Ziels, das der Client in diesem Moment zu erfüllen versucht, wird so die Applikation durch den Server mittels zur Verfügungstellung entsprechender Hyperlinks geleitet und gesteuert [7]. Der Vorteil der Verwendung von Hyperlinks liegt unter anderem in ihrer universellen Anwendbarkeit. Jede Web-Applikation kann unabhängig von ihrer Technologie Hyperlinks lesen, verwenden und verschicken [7].

- Standardmethoden:

Damit dieses Zusammenspiel der Ressourcen mittels Hyperlinks funktioniert, braucht es gewisse Standards, die von allen Teilen des Systems verstanden und implementiert werden. Dies ist die Voraussetzung dafür. Im Falle des World Wide Web sind dies die HTTP-Methoden. Analog zu einem z.B. Java-Interface definiert HTTP ein Set von Standardmethoden mit garantiertem Verhalten und eindeutigen Definitionen, das von allen Servern und Clients unterstützt wird. Die Vielfalt an Methoden, die eine einzelne Applikation benötigt, wird dabei über die Vielfalt der Ressourcen erreicht, die von den Standardmethoden aufgerufen werden. Der Methodensatz bleibt aber fix [7].

- Unterschiedliche Repräsentationen:

Die Ressourcen können in unterschiedlichen Formaten repräsentiert werden. Auch hier gibt es ein Set von Standardformaten, wie beispielsweise HTML, XML, Text oder JSON. Je nach Verwendungszweck kann ein unterschiedliches Format eingesetzt werden. Client und Server geben sich jeweils gegenseitig an, in welchem Format sie die Ressource anfordern respektive senden. Alle Komponenten, die dasselbe Format unterstützen, können so miteinander kommunizieren [7].

- Statuslose Kommunikation:

Der Server speichert keinen Sitzungsstatus über die gesamte Dauer einer Anfrage eines Clients hinweg. Entweder wird der Status auf Seite des Clients gespeichert, oder der Server wandelt den Status in eine Ressource um, die mit einem Link aufgerufen werden und so beispielsweise auch versendet werden kann. Damit wird die Kopplung von Client und Server weiter verringert. Zwei Abfragen eines Client müssen nicht von derselben Serverinstanz verarbeitet werden und der Server könnte in der Zwischenzeit auch heruntergefahren, gewartet und wieder hochgefahren werden, ohne dass die Kommunikation Informationen verliert. Die Skalierbarkeit der Anwendungen nimmt dadurch zu [7].

Für die in dieser Arbeit programmierte API werden die REST-Prinzipien vor allem bei der Gestaltung der Endpoints berücksichtigt. Ansonsten setzt die API als API für einen Web-Service der auf HTTP basiert, die meisten REST-Prinzipien bereits automatisch um.

5

Programmierung und Dokumentation der API

5.1. Aufsetzen eines Spring Boot-Projekts	33
5.1.1. Spring Initializr	33
5.2. Quellcode	35
5.2.1. Entities	36
5.2.2. Repositories	40
5.2.3. Controller	42
5.2.4. Data Templates und Config	46
5.2.5. Main-Klasse	47
5.3. Datenbank	48
5.4. Dokumentation mit Swagger	49

In diesem Kapitel wird die in dieser Arbeit erstellte API dargestellt und erläutert. Zuerst wird das Aufsetzen eines Spring Boot-Projekts mit Maven beschrieben, dann wird der Quellcode der API erklärt und schliesslich wird die Dokumentation der API mit Swagger beschrieben.

5.1. Aufsetzen eines Spring Boot-Projekts

5.1.1. Spring Initializr

Ein Spring-Projekt kann sehr einfach und schnell mit Hilfe des Onlinetools „Spring Initializr“ (<https://start.spring.io/>) erstellt werden. Auf dieser Webseite kann man zuerst das gewünschte Built-Tool (Maven oder Gradle) wählen, dann die Programmiersprache (Java, Kotlin oder Groovy), als nächstes die Version von Spring Boot - es wird also davon ausgegangen, dass man Spring Boot nutzt - und schliesslich definiert man einen Projektnamen und einen Namen für das Paket für das Projekt. Dann kann man die gewünschten Libraries und Frameworks, die man für sein Projekt braucht, anwählen. Abbildung 5.1. zeigt die Benutzeroberfläche des Spring Initializrs.

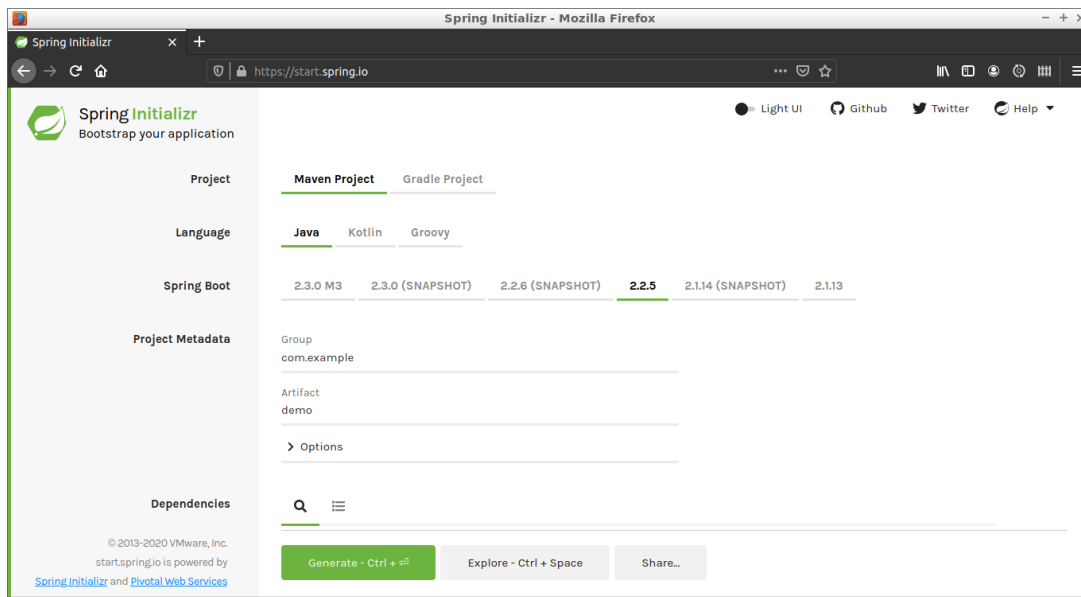


Abbildung 5.1.: Spring Initializr Onlinetool

Dann kann man das Projekt als Jar-File und Maven- respektive Gradleprojekt auf seinen Computer herunterladen und in seine IDE importieren. Dort findet man nun eine fertig eingerichtete Projektstruktur, in welcher sogar bereits die Main-Klasse erstellt ist. Für dieses Projekt wurden folgende Einstellungen verwendet:

- Project: Maven Project
- Language: Java
- Spring Boot: 2.2.0
- Project Metadata:
 - Group: ch.keilestats
 - Artifact: api
- Dependencies
 - spring-boot-starter-web
 - spring-boot-starter-data-jpa
 - spring-boot-devtools
 - spring-boot-starter-test
 - h2

Die Dependency „spring-boot-starter-web“ enthält das Spring MVC Framework, ein Logging Framework, das Spring Core-Framework, sowie ein Validierungsframework. Spring-boot-starter-JPA enthält JPA, mit Hibernate eine Standardimplementation von JPA sowie eine automatische Konfiguration dieser Komponenten [35]. Spring-boot-devtools ist eine Library zur Unterstützung bei der Entwicklung. Beim Abspeichern von Änderungen im Quellcode wird der Server beispielsweise automatisch neu gestartet, wenn diese Dependency aktiviert ist, was sehr praktisch ist. Spring-boot-starter-test ist das Framework zum Testen der Applikation. H2 ist eine In-Memory-Datenbank, die zum Entwickeln sehr gut geeignet ist. Durch die Spring Boot Autokonfiguration kann ohne Probleme später die Dependency einer anderen Datenbank, beispielsweise PostgreSQL, hinzugefügt werden und mit sehr wenig Aufwand (einzig die Authentifikation für die Datenbank muss konfiguriert werden) in Betrieb genommen werden.

Später wurden für dieses Projekt zusätzlich folgende Dependencies hinzugefügt:

- Jackson API (jackson-core und jackson-databind), um Objekte in JSON umzuwandeln und umgekehrt.
- Swagger2 und Swagger-ui, für die Generierung der Swagger-Interfaces zum Dokumentieren und Testen der API.

Diese können vom Maven Online-Repository bezogen werden und müssen nur ins pom.xml-File eingefügt werden. Somit stehen sie bereits für das Projekt zur Verfügung. Das pom.xml-File ist im Anhang dargestellt.

5.2. Quellcode

Projektstruktur in Eclipse

Die Ordnerstruktur des Projekts in der IDE besteht aus fünf Packages: Das erste, „entities“, beinhaltet die Klassen für die Entitäten, die in der Datenbank abgespeichert werden, das zweite, „repositories“, enthält die Repository-Interfaces für die CRUD-Operationen auf diesen Entitäten, das dritte, „controller“, die Controller, welche die Requests an die Endpoints verarbeiten, ein viertes, „datatemplates“, enthält Klassen für die Formatierung der Daten für ihre Über- und Ausgabe an die Controller-Methoden und ein fünftes Package, „config“, enthält Konfigurationsinformationen für die Generierung der Swagger2-Dokumentation [47] der API. Abbildung 5.2. zeigt die Ordnerstruktur und die Klassen in der IDE.

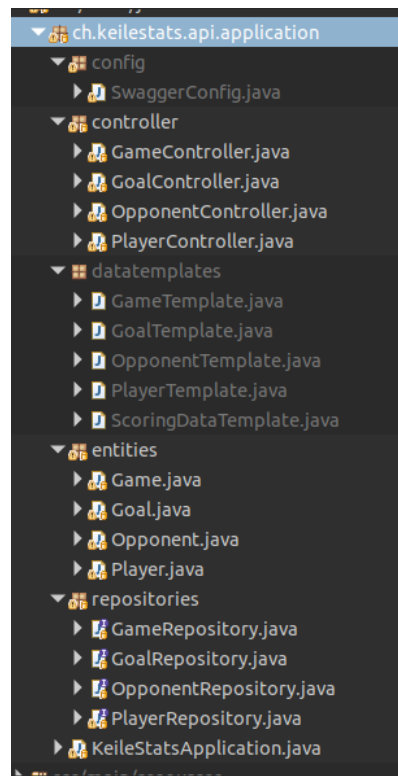


Abbildung 5.2.: Ordnerstruktur, Packages, Interfaces und Klassen

Auf derselben Ordnerhierarchieebene wie die Packages ist die Main-Klasse, welche die Applikation startet, die Konfiguration liest und die Beans kreiert. Diese Struktur der Packages und vor allem, dass die Main-Klasse auf derselben Ebene wie die Packages angeordnet ist, ist in Spring Boot zwingend, damit die Spring Boot das Projekt automatisch konfigurieren kann. Im folgenden werden die Packages und ihre Inhalte genauer beschrieben.

5.2.1. Entities

Das Package „entities“ enthält die Klassen, die die Entitäten in der Datenbank repräsentieren. Im Falle der hier programmierten API sind dies die Klassen „Game“, „Goal“, „Player“ und „Opponent“. In Listing 5.1. ist exemplarisch die Klasse „Game“ dargestellt, die im Folgenden erläutert wird. Die übrigen Entitäts-Klassen verwenden dieselben Elemente und haben dieselbe Struktur.

```
1 package ch.keilestats.api.application.entities;  
2  
3 // imports omitted  
4  
5 @Entity  
6 /*
```

```

7  * @Entity = Annotation that marks the Class to JPA as a persistent Entity and
8  * indicates it to the framework as a bean to instantiate for the application.
9  */
10 public class Game {
11
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     @Column(nullable = false)
15     private Long gameId;
16
17     private String gameDate;
18
19     // Cascade attribute defines how changes on one side of the association are
20     // "cascaded" to the other side.
21     @OneToMany(mappedBy = "game", cascade = CascadeType.REMOVE)
22     private List<Goal> goalsKeile = new ArrayList<>();
23
24     private Integer nbGoalsKeile;
25     private Integer nbGoalsOpponent;
26
27     @ManyToOne
28     @JoinColumn(name = "OPPONENT_ID") // marks and names column, where the foreign key
        of the related entity is saved
29     private Opponent opponent;
30
31     @ManyToMany
32     // Definition of the columns of the new table emerging from the many-to-many
33     // relationship
34     @JoinTable(joinColumns = @JoinColumn(name = "game_id"), inverseJoinColumns =
        @JoinColumn(name = "player_id"))
35     @JsonIgnoreProperties("games")
36     private List<Player> players = new ArrayList<>();
37
38     public Game() {} // Empty constructor required by the framework
39
40     public Game(String gameDate, Opponent opponent, Integer nbGoalsKeile, Integer
        goalsOpponent, List<Player> players,
41         List<Goal> goalsKeile) {
42
43         this.gameDate = gameDate;
44         this.goalsKeile = goalsKeile;
45         this.nbGoalsKeile = nbGoalsKeile;
46         this.nbGoalsOpponent = goalsOpponent;
47         this.opponent = opponent;
48         this.players = players;
49     }
50
51     public Long getGameId() {
52         return gameId;
53     }
54
55     public void setGameId(Long id) {
56         this.gameId = id;
57     }
58
59     @JsonManagedReference(value = "GoalinGame")
60     /*
61     * Annotation used to indicate that annotated property is part of two-way

```

```

62     * linkage between fields; and that its role is "parent" (or "forward") link.
63     * Necessary to break infinite loop at serialisation and deserialisation
64     */
65     public List<Goal> getGoalsKeile() {
66         return goalsKeile;
67     }
68
69     public void setGoalsKeile(List<Goal> goals) {
70         this.goalsKeile = goals;
71     }
72
73     @JsonManagedReference(value = "opponentInGame")
74     public Opponent getOpponent() {
75         return opponent;
76     }
77
78     public void setOpponent(Opponent opponent) {
79         this.opponent = opponent;
80     }
81
82     @JsonManagedReference
83     public List<Player> getPlayers() {
84         return players;
85     }
86
87     public void setPlayers(List<Player> players) {
88         this.players = players;
89     }
90
91     // some more getters and setters
92 }
93 // to string and equals methods omitted for reasons of brevity

```

Listing 5.1: Klasse Game

Annotation @Entity

Mit der @Entity-Annotation auf Klassenlevel (siehe Listing 5.1., Zeile 5) wird markiert, dass die Klasse eine persistente Entität ist. Für sie wird eine Tabelle in der Datenbank erstellt. Die meisten Felder der Klasse entsprechen Tabellenspalten, einige Felder sind durch Relationen mit anderen Entitäten verbunden, dort wird durch die Konfiguration festgelegt, wie die Beziehung genau auf die Tabellen abgebildet wird (siehe Erläuterung der Annotationen unten). Der Name der Klasse wird per Defaulteinstellung zum Tabellennamen, sofern JPA nicht anders konfiguriert wird, oder mittels „name“-Attribut der Annotation @Table der Name anders festgelegt wird [12].

Annotationen @Id, @GeneratedValue und @Column

Mit der @Id-Annotation wird ein Feld als Primärschlüssel der Entität markiert (siehe Listing 5.1., Zeile 12). Dieses muss einem bestimmten Set an Typen und Klassen entsprechen, um diese Funktion erfüllen zu können [26].

Mit der Annotation @Column (siehe Listing 5.1., Zeile 14) können die Eigenschaften des Attributs der Tabelle präzisiert werden. Zum Beispiel kann der Name der Tabelleneigenschaft vom Default-Wert abweichend spezifiziert werden, oder es kann angegeben werden, ob die gespeicherten Werte leer sein dürfen (d.h. ob sie den Wert „null“ annehmen dürfen oder nicht).

Mit der Annotation `@GeneratedValue` (siehe Listing 5.1., Zeile 13) wird die Strategie eingestellt, mit der Hibernate den Primärschlüssel generiert. Hier wurde „sequence“ verwendet. Das heisst, es wird eine Sequenz, welche von der Datenbank generiert wird, zur Erzeugung der Primärschlüssel eingesetzt [36].

Annotationen `@OneToMany`, `@ManyToOne` und `@ManyToMany`

Das Feld „goalsKeile“, das eine Liste von Objekten der Klasse `Goal` enthält, repräsentiert die Tore, welche der HC Keile in einem Spiel erzielt (siehe Listing 5.1., Zeile 22). Das Feld ist also mit der Entität `Goal` assoziiert und zwar mit einer One-to-many-Assoziation: die Liste enthält null, eines oder mehrere Tore, mehrere Tore können ein und demselben Spiel angehören und ein Tor gehört zu genau einem Spiel.

Die `@OneToMany`-Annotation (siehe Listing 5.1., Zeile 21) deklariert diese Beziehung. Ergänzend ist das andere Ende der Beziehung, das Feld „game“ in der Klasse `Goal`, mit einer `@ManyToOne`-Annotation markiert. Das Feld enthält die Information, welchem Spiel das Tor zuzuordnen ist. Mit dem Attribut „mappedBy“ der `@OneToMany`-Annotation wird der Persistenzschicht mitgeteilt, dass die Beziehung in diesem Feld abgebildet und mittels Fremdschlüssel in diesem Feld in der Datenbank gespeichert werden soll. Das Feld „game“ in der Tabelle `Goal` somit wird als Träger der Information über die Beziehung festgelegt.

Das Attribut „cascade“ (siehe Listing 5.1., Zeile 21) bestimmt, wie Daten, die in der `Game`-Klasse gespeichert werden, auch in den assoziierten Klassen - respektive auf Datenbankebene in den entsprechenden Tabellen - gespeichert, gelöscht oder verändert werden. Hier ist „CascadeType.REMOVE“ eingestellt, d.h. Delete-Operationen werden auch auf die im Objekt-Graph angehängten Entitäten und Objekte übertragen. Wenn man z.B. ein Spiel löscht, werden auch die Tore, die in diesem Spiel gefallen sind, aus der „Goal“-Tabelle gelöscht.

Die Annotation `@ManyToMany` markiert eine Many-to-Many-Beziehung. In diesem Fall hat ein Spiel eine Liste von Spielern (siehe Listing 5.1., Zeile 36) und jeder Spieler hat eine Liste von Spielen. In der `@JoinTable`-Annotation wird die zusätzliche Tabelle, welche diese Informationen enthält, definiert. Die beiden Spalten, welche Paare von Spiel-ID und Spieler-ID enthalten, werden deklariert und es wird ein Name für sie angegeben. In der `Player`-Klasse ist zusätzlich mit dem „mappedBy = players“-Attribut die `Game`-Klasse als „owner“ der Beziehung deklariert. Das heisst, dass die Konfigurationsinformationen in dieser Klasse massgebend sind. Dies ist in bidirektionalen Beziehungen nötig zu definieren [27].

Annotationen `@JsonIgnoreProperties`, `@JsonManagedReference` und `@JsonBackReference`

Wenn zwei oder mehrere Entitäten durch eine Beziehung verbunden sind und so gegenseitig aufeinander Bezug nehmen, kann ein infinites Loop entstehen. Zum Beispiel ruft die Entität „Spiel“ eine Liste an Spielern auf, diese rufen ihre Listen an Spielen auf, welche wiederum ihre Listen an Spielern aufrufen und so weiter. Um diese Loops abubrechen, braucht es Konfigurationsinformationen, die das Framework anweisen, bei Serialisierung und Deserialisierung nur den Aufruf der einen Seite der Beziehung zu tätigen und dann zu stoppen.

Für diesen Zweck können die Annotationen `@JsonManagedReference` und `@JsonBackReference` sowie die Annotation `@JsonIgnoreProperties` eingesetzt werden. Die ersten beiden werden über den Getter-Methoden (siehe Listing 5.1., Zeilen 59, 73 und 82) angebracht, `@JsonManagedReference` auf der einen Seite und `@JsonBackReference` auf der anderen Seite der Beziehung. Dadurch wird der infinite Loop nach Aufruf der mit `@JsonManagedReference` annotierten Seite gestoppt.

Dasselbe wird erreicht, wenn man mit `@JsonIgnoreProperties` angibt, welche Felder bei der Serialisierung und Deserialisierung ignoriert werden sollen (siehe Listing 5.1., Zeilen 35).

Schliesslich benötigt JPA einen leeren Konstruktor (siehe Listing 5.1., Zeile 38), um die Entitäten zu instantiieren. [23]

5.2.2. Repositories

Das Package „repositories“ enthält die Repository-Interfaces, welche Manipulationen auf der Datenbank ermöglichen. Das Repository-Interface aus Spring Data respektive eines seiner Subinterfaces liefern in Spring die Methodendefinitionen für diese Manipulationen. Abbildung 5.3. zeigt einen Ausschnitt aus der Interface-Hierarchie des Repository-Interfaces. Für die hier programmierte API wurde das `JpaRepository` verwendet (siehe Listing 5.2.), welches eine JPA-spezifische Erweiterung des Repository-Interfaces und von Subinterfaces davon darstellt [31].

```
org.springframework.data.jpa.repository

Interface JpaRepository<T, ID>

All Superinterfaces:
CrudRepository<T, ID>, PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T>, Repository<T, ID>

All Known Subinterfaces:
JpaRepositoryImplementation<T, ID>

All Known Implementing Classes:
QuerydslJpaRepository, SimpleJpaRepository
```

```
@NoRepositoryBean
public interface JpaRepository<T, ID>
extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T>

JPA specific extension of Repository.
```

Abbildung 5.3.: Sub- und Superinterfaces von `JpaRepository`, Javadoc

Ab Hibernate 5.4 müssen die Interfaces nicht mehr implementiert werden [6]. Die Methodenaufrufe werden mit Hilfe einer `MethodInterceptor`-Klasse abgefangen. Sie schaut zuerst, ob es eine Implementierung der Methode im Quellcode gibt, falls nicht, versucht sie, die nötigen SQL-Abfragen aus dem Code zu ermitteln, entweder aus im Code explizit formulierten Queries oder über den Namen der Methode. Falls das nicht geht, greift die Klasse auf eine von Spring Data zur Verfügung gestellte Basisklasse zurück, die eine Standardimplementierung der Interface-Methode zur Verfügung stellt. Es findet dabei im Hintergrund keine Codegenerierung oder Bytecode-Manipulation statt [6, S. 221] [23].

```
1 package ch.keilestats.app.repositories;
2
```

```
3 import java.util.Optional;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6 import ch.keilestats.app.entities.*;
7
8 @Repository
9 public interface OpponentRepository extends JpaRepository<Opponent, Long> {
10
11     Optional<Opponent> findByOpponentName(String opponentName);
12
13 }
```

Listing 5.2: Interface JpaRepository

Listing 5.2. zeigt exemplarisch die hier verwendete Deklaration für die Zugriffe auf die Entität „Opponent“. Für die übrigen Klassen der HC Keile-API sieht die Deklaration sehr ähnlich aus. Es wird ein Interface deklariert, das, wie erklärt, eines der Repository-Subinterfaces erweitert und das den Typ der Entität, auf die damit zugegriffen werden soll, deklariert sowie den Typ des Primärschlüssels, der in der Entität verwendet wird (siehe Listing 5.2., Zeile 9) [45]. Damit stehen nun Methoden wie „save(entity)“, „delete(entity)“, „findAll()“, oder „findById(id)“ zum Speichern, Löschen, Laden aller Einträge oder Laden eines einzelnen Eintrags zur Verfügung sowie weitere Methoden.

Klasse Optional<T>

Die Klasse „Optional<T>“ aus der java.util-Library repräsentiert ein Containerobjekt, das entweder einen von null verschiedenen Wert enthält, oder leer ist. Dies kann mit der Methode „isPresent()“, die true zurückgibt, wenn das Optional-Objekt nicht leer ist und false, wenn nicht, geprüft werden. Mit „get()“ kann auf den Wert zugegriffen werden [28].

Die Klasse wird hier eingesetzt, um die Resultate von Abfragen auf der Datenbank entgegen zu nehmen (siehe Listing 5.2., Zeile 11, sowie Listing 5.3., Zeilen 31 und 33). Dadurch kann der Fall, dass nichts in der Datenbank gefunden wird, gehandhabt werden, ohne dass man in Probleme mit null-pointer-exceptions gerät.

Definition eigener Queries

Zeile 11 in Listing 5.2. zeigt ein Beispiel einer selber definierten Methode, die über den Methodennamen automatisch von Spring implementiert wird. „OpponentName“ ist ein Feld der Klasse und Entität „Opponent“. Mittels des Standardausdrucks „findBy“ werden die Einträge aus der Opponent-Tabelle der Datenbank gesucht, welche dem Methodenparameter entsprechen. Es können dabei noch deutliche komplexere Queries über den Methodennamen definiert werden. Die Spring Data Referenz-Dokumentation enthält eine Liste der unterstützten Schlüsselwörter für die Definition von Queries über den Methodennamen [44]. In Tabelle 5.1. ist diese Liste dargestellt.

Keyword	Sample	JPQL snippet
And	findByLastNameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastNameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Is, Equals	findByFirstname, findByFirstnames, findByFirstnameEquals	... where x.firstname = ?1
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	... where x.age <= ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	... where x.age >= ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull, Null	findByAge(Is)Null	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> ages)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	... where UPPER(x.firstname) = UPPER(?1)

Tabelle 5.1.: Unterstützte Keywords in JPA für Query-Definition über den Methodennamen

Die Java Persistence Query Language (JPQL) kann in JPA gebraucht werden, um eigene Queries zu definieren. Sie ist SQL sehr ähnlich und wird von der JPA-Implementation in den Dialekt der entsprechenden verwendeten Datenbank umgewandelt. Somit kann die Datenbank geändert werden, ohne dass es Probleme gibt mit Details in der Formulierung der Queries. Mit der Annotation `@Query` über dem Methodennamen kann die JPQL-Query direkt über der definierten Methode formuliert werden [35].

Persistence Context

Hibernate bewerkstelligt die Datenbanktransaktionen automatisch im Hintergrund. Die Umgebung, in der Objekte instanziiert sowie Daten und Operationen geladen werden, um die gewünschten Manipulationen auf der Datenbank vorzunehmen, heisst in JPA „Persistence Context“ [30]. In Hibernate liefert das „Session“-Interface den Zugang zum Persistence Context [34]. Die Session wird am Anfang der Transaktion (normalerweise bei einem Methodenaufruf einer der Repository-Methoden) automatisch kreiert und nach dem Ende aller Datenbankoperationen wieder aufgelöst [35].

5.2.3. Controller

Das package „controller“ enthält die Controller-Klassen, die auf die Repositories und Entitäten als Dependencies zugreifen. Im Model-View-Controller (MVC)-Schema ist die Funktion des Controllers, HTTP-Requests, die auf eine spezifische Domain geschickt werden, pro Domain abzuarbeiten. Der Controller implementiert, wie die einzelnen Requests verarbeitet werden sollen. Listing 5.3. zeigt exemplarisch den Controller für die Abfragen auf die „Player“-Ressource. Die Controller für die übrigen Entitäten sind analog aufgebaut und verwenden die gleichen Klassen und Annotationen.

```

1 package ch.keilestats.api.application.controller;
2
3 //imports omitted for brevity
4
```

```
5 @RestController
6 /*
7  * A convenience annotation that is itself annotated with @Controller
8  * and @ResponseBody. @Controller is a specification of @Component to indicate
9  * to the framework that the container should instantiate the class as a bean
10 */
11 @RequestMapping("/api")
12 /*
13  * Annotation for mapping web requests onto methods in request-handling classes
14  * with flexible method signatures
15 */
16 public class PlayerController {
17
18     @Autowired /* Annotation to tell the framework to inject this dependency */
19     private PlayerRepository playerRepository;
20
21     // Return list of all players
22     @GetMapping("/players")
23     public ResponseEntity<Object> getAllPlayers() {
24
25         return ResponseEntity.status(HttpStatus.OK).body(playerRepository.findAll());
26     }
27
28     @DeleteMapping("/players/{playerId}")
29     public ResponseEntity<Object> deletePlayer(@PathVariable("playerId") Long playerId)
30     {
31
32         if (playerRepository.findById(playerId).isPresent()) {
33
34             Player playerToBeDeleted = playerRepository.findById(playerId).get();
35
36             if (playerToBeDeleted.getGames().isEmpty()) {
37                 playerRepository.deleteById(playerId);
38                 return new ResponseEntity<>("Player deleted..", HttpStatus.OK);
39             } else
40                 return new ResponseEntity<>("Player with id: " + playerId + " has a non-
41                     empty List of games "
42                     + "and therefore cannot be deleted", HttpStatus.BAD_REQUEST);
43         }
44
45         return new ResponseEntity<>("Player with id " + playerId + " not found",
46             HttpStatus.BAD_REQUEST);
47     }
48
49     // Return values of one Player
50     @GetMapping("/players/{playerId}")
51     public ResponseEntity<Object> getPlayerById(@PathVariable("playerId") Long
52         playerId) {
53
54         Optional<Player> playerOptional = playerRepository.findById(playerId);
55
56         if (playerOptional.isPresent()) {
57             return ResponseEntity.status(HttpStatus.OK).body(playerOptional.get());
58         } else {
59             return ResponseEntity.status(HttpStatus.NOT_FOUND).body("player with id " +
60                 playerId + " not found");
61         }
62     }
63 }
```

```
58 // Returning a ResponseEntity with a header containing the URL of the created
59 // resource
60 @PostMapping(path = "/players")
61 public ResponseEntity<Object> addPlayer(@RequestBody PlayerTemplate playerTemplate)
62 {
63
64     Player player = new Player();
65
66     player.setFirstname(playerTemplate.getFirstname());
67     player.setLastname(playerTemplate.getLastname());
68     player.setAddress(playerTemplate.getAddress());
69     player.setPosition(playerTemplate.getPosition());
70     player.setEmail(playerTemplate.getEmail());
71     player.setPhone(playerTemplate.getPhone());
72
73     if (playerRepository.findAll().contains(player)) {
74         return new ResponseEntity<>(
75             "Player with name \"\" + player.getFirstname() + \" \" + player.
76               lastname() + \"\" already exists",
77             HttpStatus.BAD_REQUEST);
78     }
79
80     Player savedPlayer = playerRepository.save(player);
81
82     URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{
83         playerId}")
84         .buildAndExpand(savedPlayer.getPlayerId()).toUri();
85
86     return ResponseEntity.created(location).body("player created");
87 }
88 //...
89 //more endpoints omitted for reasons of brevity
```

Listing 5.3: Klasse PlayerController

Annotation @RestController

Die @RestController-Annotation (siehe Listing 5.3., Zeile 5) ist eine Abkürzung für die Annotationen @Controller und @ResponseBody [33]. Sie hat zwei Hauptzwecke. Erstens wird mit ihr die Klasse als Component für das Component Scanning markiert, d.h. sie wird vom Container als Bean der Applikation instantiiert. Zweitens sagt sie Spring, dass die Rückgabewerte aller Handler-Methoden der Klasse direkt in den Response-Body der HTTP-Response geschrieben werden sollen, anstatt dass sie in ein Model gespiesen und eine HTML-View daraus generiert werden soll [9]. Dies ist der Unterschied zu einem traditionellen Spring MVC Controller, der üblicherweise Views zurückgibt [45].

Annotation @RequestMapping

Die Annotation @RequestMapping (siehe Listing 5.3., Zeile 11) auf Klassenlevel, zusammen mit den seit Spring 4.3. eingeführten spezifischen @PostMapping, @GetMapping, @PutMapping, @DeleteMapping-Annotationen etc. (siehe Listing 5.3., Zeilen 22, 28, 37 und 61) auf

Methoden-Level, bewirken, dass die entsprechenden HTTP-Requests auf die entsprechenden Methoden gemappt werden. Es ist gute Praxis, auf Klassenlevel bei den Controller-Klassen die `@RequestMapping`-Annotation zu benutzen und dabei den Basis-URI-Pfad zu definieren, und dann bei den Methoden, welche bei den einzelnen Requests aufgerufen werden, zu spezifizieren, um welche Art von Methode und welche URI es sich handelt [9, S. 35]. Dies wird hier so umgesetzt.

Annotation `@RequestBody`

Die `@RequestBody`-Annotation (siehe Listing 5.3., Zeile 61) sagt Spring, dass die Daten aus dem Body des HTTP-Request in das Objekt, das dem Controller übergeben wird, konvertiert werden sollen. Ohne diese Annotation geht Spring davon aus, dass es die Parameter des HTTP-Requests in das Objekt umwandeln soll [9][S. 146].

Die `@PathVariable`-Annotation (siehe Listing 5.3., Zeile 29) gibt die Anweisung, dass die Variable in die Request-URI eingebunden werden soll, sie mappt den Parameter der Methode auf eine URI-Template-Variable [29].

Klasse `ResponseEntity<T>`

Die Klasse `ResponseEntity` (siehe Listing 5.3., z.B. Zeilen 23, 25 und 43) erweitert die Klasse `HttpEntity`, welche eine `Http-Response` oder `Http-Request-Entität` repräsentiert [32]. Mit Hilfe dieser Klasse kann der HTTP-Status bestimmt werden, der in der Antwort zurückgegeben werden soll, je nach Verlauf der Verarbeitung, und es kann festgelegt werden, was im Response Body enthalten sein soll.

Bei erfolgreichem Request werden bei der HC Keile API die geforderten oder abgespeicherten Daten im JSON-Format im Request Body zurückgegeben sowie ein HTTP-Status, meist „200: OK“ oder „201: CREATED“ (siehe Listing 5.3., Zeilen 25, 37, 53 und 85). Wenn eine neue Ressource angelegt wird bei POST-Requests, wird im Header der `Http-Response` mit Hilfe der „URI“-Klasse der `Java.net`-Library sowie der `ServletUriComponentsBuilder`-Klasse des Spring Frameworks die URI der neu angelegten Ressource zurückgegeben (siehe Listing 5.3., Zeile 82).

Bei Anfragen, die zu einem Fehler führen, wie beispielsweise, wenn bei einer GET-Methode eine ID übergeben wird, die nicht existiert, oder wenn bei der POST-Methode ein Spieler gespeichert werden soll, der den gleichen Namen und Vornamen hat, wie einer, der bereits in der Datenbank vorhanden ist, kann im Request-Body eine Fehlermeldung ausgegeben werden, in welcher der aufgetretene Fehler erklärt wird. Dazu wird meistens der HTTP-Status „400: BAD REQUEST“ als Response-Status definiert (siehe Listing 5.3., Zeilen 73-76).

In der `PlayerController`-Klasse kann zum Beispiel ein Spieler, der bereits Spiele gespielt hat und in einem Datenbankeintrag der Entität „Game“ in der Spielerliste dieses Spiels eingetragen ist, nicht gelöscht werden, solange dies der Fall ist. Wird versucht, den Spieler trotzdem zu löschen, wird die Meldung: „Player with id x has a non-empty list of games and therefore cannot be deleted“ mit dem Status „400: BAD REQUEST“ zurückgegeben (siehe Listing 5.3., Zeilen 35-40). Bei der Speicherung eines neuen Spielers wird, wie beschrieben, geprüft, ob ein Spieler mit demselben Namen und Vornamen bereits in der Datenbank existiert, falls ja, muss ein anderer Name gewählt werden. In den meisten Fällen muss kontrolliert werden, ob die Ressource existiert, um null-pointer-exceptions zu vermeiden.

Jackson - Java JSON API

Die Umwandlung der Objekte von Java in JSON bei der Serialisierung und umgekehrt geschieht dabei im Hintergrund automatisch mit Hilfe der Java JSON API mit dem Namen „Jackson“. Dazu

müssen die entsprechenden Jackson-Dependencies im pom.xml-file der Applikation spezifiziert und eingebunden werden. Listing 5.4. zeigt die Dependencies im pom.xml-File.

```
1 //...
2     <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core
3         -->
4     <dependency>
5         <groupId>com.fasterxml.jackson.core</groupId>
6         <artifactId>jackson-core</artifactId>
7     </dependency>
8     <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
9         databind -->
10    <dependency>
11        <groupId>com.fasterxml.jackson.core</groupId>
12        <artifactId>jackson-databind</artifactId>
13    </dependency>
14 //...
```

Listing 5.4: pom.xml Jackson Dependencies

Nebst der PlayerController-Klasse enthält das package „controller“ die weiteren Controllerklassen (GameController, OpponentController und GoalController) der Applikation für das Handling der Abfragen auf die restlichen Entitäten „Game“, „Goal“ und „Opponent“.

5.2.4. Data Templates und Config

Im Package „datatemplates“ sind Klassen definiert, mit denen den Controller-Methoden Daten übergeben werden können, ohne dass ein vollständiges Objekt einer Entität, mitsamt allen Feldern und Listen, welche für die Datenübergabe oft nicht nötig sind, übergeben werden muss. Dies dient auch der Erleichterung des Testens der API im Swagger-Interface, da die JSON-Objekte, die den HTTP-Methoden übergeben werden oder von diesen zurückgegeben werden, so viel reduzierter und übersichtlicher sind. Listing 5.5. zeigt exemplarisch das Datentemplate für die Entität „Goal“ :

```
1 package ch.keilestats.app.datatemplates;
2
3 /*Class to help to pass data to the POST and PUT-methods of the goal controller.
4  * Template for collecting, saving and presenting goal data*/
5 public class GoalTemplate {
6
7     Long goalScorerId;
8     Long firstAssistantId;
9     Long secondAssistantId;
10
11     public GoalTemplate() {
12     }
13
14 // getters and setters and toString-Method omitted
15
16 }
```

Listing 5.5: Klasse GoalTemplate

Im package „config“ ist einzig die Konfigurationsdatei für das Swagger-Interface zur Dokumentation und zum Testen der API abgelegt. Die Swagger-Dokumentation der API ist in Kapitel 5.3. dargestellt.

5.2.5. Main-Klasse

Die Main-Klasse der Applikation heisst „KeileStatsApplication“. Sie umfasst eine main-Methode, in welcher einzig die statische Methode „SpringApplication.run(KeileStatsApplication.class, args)“ aufgerufen wird. Mit dieser Methode wird die Applikation gebootstrapt. Die zweite wichtige Komponente ist die Annotation @SpringBootApplication. Sie ist eine Composite-Annotation, die weitere Annotationen umfasst:

@SpringBootConfiguration, die eine spezialisierte Form der @Configuration-Annotation ist, gibt an, dass in dieser Klasse Java-basierte Konfigurationsinformationen enthält [9, S. 15]. @EnableAutoConfiguration ermöglicht Spring Boot, die Applikation automatisch zu konfigurieren. @ComponentScan ermöglicht Component scanning. Das heisst, dass andere Klassen mit @Component-Annotationen als Komponenten der Applikation im Spring Application Context registriert werden. [9, S. 15]

Die Annotation @Component einer Klasse sagt dem Framework, dass diese Klasse eine Bean ist, die instantiiert werden soll. Hier werden mit @RestController, @Entity und @Repository spezialisierte Formen der @Component-Annotation eingesetzt.

```
1 package ch.keilestats.app;
2
3 //imports omitted
4
5 /* Annotation @SpringBootApplication: Indicates a configuration class that
6  * declares one or more @Bean methods and also triggers auto-configuration,
7  * component scanning, and configuration properties scanning. This is a convenience
8  * annotation that is equivalent to declaring @Configuration,
9  * @EnableAutoConfiguration, @ComponentScan.
10 */
11 @SpringBootApplication
12 /*
13  * Class SpringApplication bootstraps the Application: -creates an
14  * ApplicationContext instance -registers a CommandLinePropertySource to expose
15  * command line arguments as spring properties -refresh application context,
16  * loading all singleton beans -Trigger Any command line runner beans
17  */
18 public class KeileStatsApplication implements CommandLineRunner {
19
20     Logger logger = LoggerFactory.getLogger(getClass());
21
22     // Repositories used for setUpData()-Method in order to create sample-Data to test the
23     API
24     @Autowired
25     GameRepository gameRepository;
26     @Autowired
27     PlayerRepository playerRepository;
28     @Autowired
29     OpponentRepository opponentRepository;
30     @Autowired
31     GoalRepository goalRepository;
```

```
32 public static void main(String[] args) {
33
34     SpringApplication.run(KeileStatsApplication.class, args);
35 }
36
37 @Override
38 public void run(String... args) throws Exception {
39     // TODO Auto-generated method stub
40     setUpData();
41 }
42
43 /* Create some Data for testing purpose and save in Database */
44 public void setUpData() {
45
46 //method body omitted
47
48 }
49 }
```

Listing 5.6: Main-Klasse KeileStatsApplication

Der Application Context, der von der Methode „run“ der Klasse SpringApplication zurückgegeben wird, managt das Instantiieren der Beans und die Injektion der Dependencies [35].

Wenn die Applikation startet, sorgt die Spring Boot Autokonfiguration dafür, dass die Beans im Spring Application Context (dem Container) erstellt und konfiguriert werden sowie dass der eingebettete Tomcat-Server konfiguriert und gestartet wird [9, S.26].

5.3. Datenbank

Als Datenbank wird für die Entwicklung der API in dieser Arbeit eine h2 in-Memory-Datenbank verwendet. Sie kann ganz einfach durch das Hinzufügen der Dependency im pom.xml-file genutzt werden. Spring Boot konfiguriert sie automatisch. Mit der h2-Console steht eine graphische Benutzeroberfläche zur Verfügung, mit der direkt auf die Datenbank zugegriffen werden kann. Sie kann über <http://localhost:8080/h2-console/> aufgerufen werden, sobald die Applikation läuft. Abbildung 5.4. zeigt einen Screenshot der h2-console.

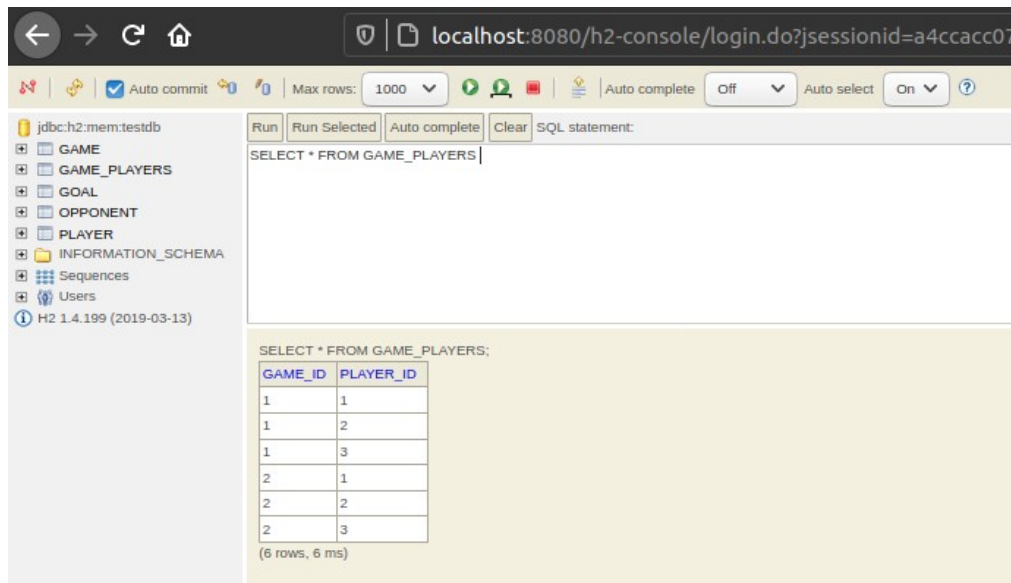


Abbildung 5.4.: h2-console

Die Datenbank kann in einer Spring Boot Applikation sehr leicht gewechselt werden. Spring Data JPA und Hibernate unterstützen alle gängigen Datenbanktechnologien. Man muss dazu nur im Maven pom.xml-File die entsprechende Dependency einfügen, Zugangscredentials (Datenbankname, Username, Passwort) angeben, falls nötig, sowie den Datenbankdialekt für Hibernate angeben, damit bessere Queries aus JPQL generiert werden können. Diese Angaben werden im application.properties-File im Order src/main/ressources in den Attributen „spring.datasource.url=*DBPortAndName*“, „spring.datasource.username=*Username*“, „spring.datasource.password=*Passwort*“ und „spring.jpa.- properties.hibernate.dialect=*DBDialect*“ gemacht.

5.4. Dokumentation mit Swagger

Swagger ist ein Tool zum Dokumentieren und Testen von API's. Mit Spring und Spring Boot kann mit Hilfe von Maven sehr einfach eine Swagger-Representation der API generiert werden. Hierfür müssen zwei Dependencies ins pom.xml-file integriert werden, die Swagger2-Dependency und die SwaggerUI-Dependency.

```

1 <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
2 <dependency>
3   <groupId>io.springfox</groupId>
4   <artifactId>springfox-swagger-ui</artifactId>
5   <version>2.9.2</version>
6 </dependency>
7
8 <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
9 <dependency>
10  <groupId>io.springfox</groupId>

```

```

11 <artifactId>springfox-swagger2</artifactId>
12 <version>2.9.2</version>
13 </dependency>

```

Listing 5.7: Maven Dependencies Swagger

Dann muss eine Konfigurationsklasse erstellt werden, die es Swagger ermöglicht, die nötigen Informationen aus dem Projekt, die vorhandenen Endpoints, zu lesen und daraus das Swagger User-Interface zu generieren.

```

1 package ch.keilestats.app.config;
2
3 //imports ommited
4
5 @EnableSwagger2
6 /*Indicates that Swagger support should be enabled.
7  * This should have an accompanying '@Configuration' annotation.
8  * Loads all required beans.
9  * */
10 @Configuration
11 /*Indicates that a class declares one or more @Bean methods and may be
12 processed by the Spring container to generate bean definitions
13 and service requests for those beans at runtime*/
14 public class SwaggerConfig {
15
16     /* Returns the Swagger2 Interface */
17     @Bean
18     public Docket keileAPI() {
19         return new Docket(DocumentationType.SWAGGER_2).select()
20             .apis(RequestHandlerSelectors.basePackage("ch.keilestats.app")).paths(
21                 regex("/app.*"))
22             .build().apiInfo(metaInfo());
23     }
24
25     // Titletext to be displayed in the Swagger interface
26     private ApiInfo metaInfo() {
27
28         ApiInfo apiInfo = new ApiInfo("Keile Stats API",
29             "API for managing " + "statistics of a just-for-fun Icehockey Team", "
30             1.0", "Terms of Service", "", "",
31             "");
32
33         return apiInfo;
34     }
35 }

```

Listing 5.8: Swagger Configuration Class

Über die URI <http://localhost8080/swagger-ui.html> kann nun auf die Swagger-Darstellung der API zugegriffen werden und diese damit getestet werden. Abbildung 5.4. zeigt das Swagger-Interface.

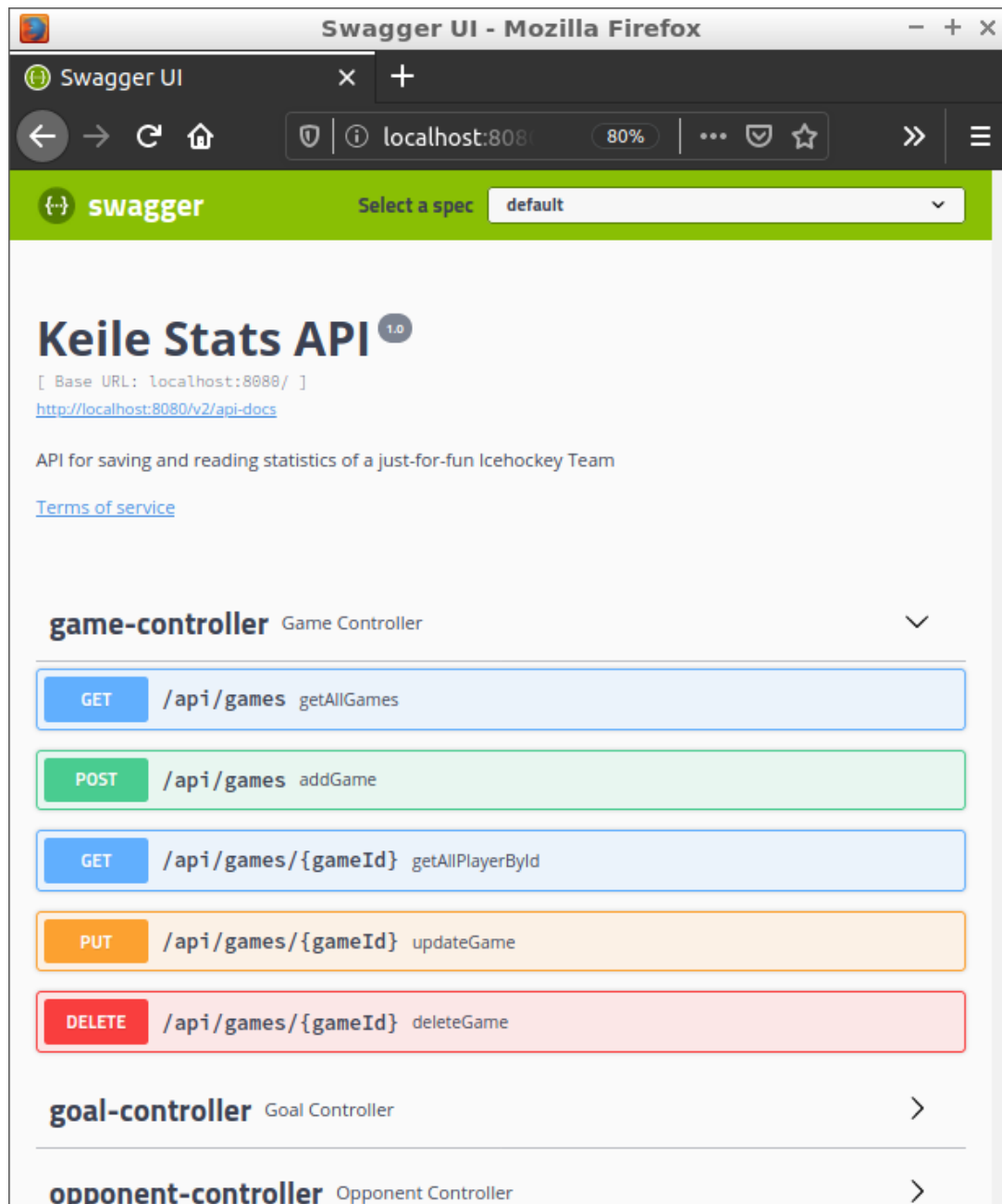


Abbildung 5.5.: Swagger Interface

6

Abschliessende Bemerkungen

6.1. Zusammenfassung

In dieser Arbeit wurde eine API nach REST-Prinzipien zum Speichern und Lesen von Daten einer Hobby-Eishockeymannschaft mit dem Java Spring-Framework und mit Spring Boot erstellt.

6.1.1. HC Keile

Ausgangspunkt war dabei die Hobby-Eishockey-Mannschaft HC Keile, die seit 2011 besteht und Freundschaftsspiele gegen andere Hobby-Eishockeyteams in der Region Freiburg (CH) durchführt. Das Team notiert bei jedem Spiel die Torschützen und Assistgeber seiner Mannschaft, sowie, welche Spieler am Spiel anwesend waren. Die Statistiken dienen nur zur Unterhaltung der Clubmitglieder, die Anzahl Spiele der Spieler werden auch verwendet, um die Eismiete pro Saison zu verrechnen und auf die Spieler aufzuteilen.

6.1.2. Anwendungsfälle der API

Die Daten der Spiele des HC Keile wurden bisher von Hand notiert, lokal in einem Excel-File gespeichert und nur einmal pro Jahr, an der Generalversammlung des Clubs, präsentiert. Die Anzahl Spiele pro Spieler wird aus mehreren Doodle-Umfrage-Files manuell für jeden Spieler zusammengezählt.

Ausgehend davon wurde hier eine API programmiert, die ein möglicher Client nutzen könnte, um diese Prozesse zu vereinfachen und effizienter zu gestalten. Das heisst, dass die Statistiken online jederzeit abgerufen werden könnten und dass sie via Webformular direkt mobil abgespeichert werden könnten, statt wie bisher, mehrmals von Hand notiert und zu Hause manuell in ein Dokument eingetragen werden müssten und lokal gespeichert werden.

Abbildung 6.1. gibt einen Überblick über die in dieser Arbeit behandelten Anwendungsfälle. Diese wurden in Kapitel zwei mittels Prozessdiagrammen im Detail ausgearbeitet und dargestellt.

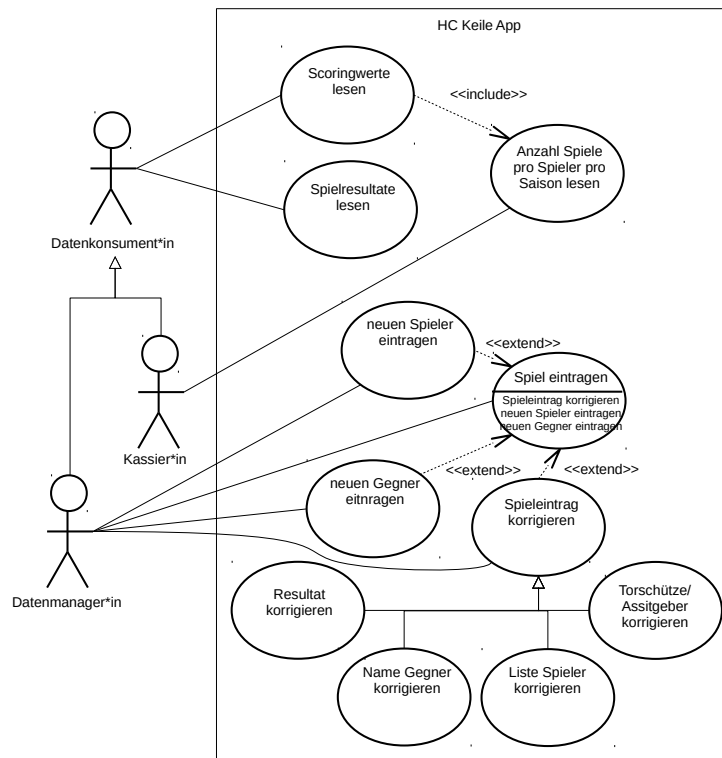


Abbildung 6.1.: Anwendungsfall-Diagramm

6.1.3. Datenmodell und Endpoints

Daraufhin wurde ein Datenmodell erstellt, das vier Entitäten („Game“, „Goal“, „Player“ und „Opponent“) umfasst, sowie nebst anderen Beziehungen eine many-to-many-Beziehung (Spieler an Spiel respektive Spiele pro Spieler), woraus sich fünf Tabellen für eine Datenbank ableiten liessen. Diese Datenbankmodellierung wurde in Kapitel drei ausgearbeitet und dargestellt.

Aus den Anwendungsfällen wurden in Kapitel 3 ebenfalls Endpoints für die API abgeleitet, mit denen die angestrebten Funktionalitäten umgesetzt werden können. Für das Design und die Auswahl der Endpoints wurden Prinzipien des Architekturstils REST berücksichtigt. Diese Prinzipien wurden in Kapitel 4.3. beschrieben. Tabelle 6.1. zeigt die 25 Endpoints, die in dieser Arbeit implementiert wurden.

Ressource	URI	Methode	Anwendungsfälle
Spieleintrag, Liste Spiele	\games	POST, GET	Spiel eintragen, Spielresultate lesen
Einzelnes Spiel	\games\{game_id}	GET, PUT, DELETE	Spieleintrag korrigieren, Spieldetails lesen
Liste Spieler an Spiel	\games\{game_id}\players	PUT	Liste Spieler korrigieren
Liste Tore	\goals	POST, GET	Tor/e eintragen
Einzelnes Tor an Spiel	\goals\{goal_id}	GET, PUT, DELETE	Torschütz*in / Assistgeber*in korrigieren
Spieler*innen	\players	GET, POST	neue Spieler*in eintragen
Einzelne Spieler*in	\players\{player_id}	GET, PUT, DELETE	Eintrag Spieler*in korrigieren, bearbeiten
Tore pro Spieler*in	\players\{player_id}\goals	GET	Scoringwerte lesen, Tore
Assists pro Spieler*in	\players\{player_id}\assists	GET	Scoringwerte lesen, Assists
Spiele pro Spieler*in	\players\{player_id}\games	GET	Scoringwerte lesen, Anzahl Spiele pro Spieler*in lesen
Topscorerliste	\players\scoringtable	GET	Scoringwerte lesen, Übersicht
Gegner	\opponents	POST, GET	neuen Gegner eintragen, Liste Gegner anzeigen
Einzelner Gegner	\opponents\{opponent_id}	PUT, GET, DELETE	Eintrag Gegner korrigieren, bearbeiten

Tabelle 6.1.: Endpoints für HC Keile-Applikation

6.1.4. Spring Framework, Spring Boot und ORM

Die API wurde in Java programmiert. Dazu wurde das Java Spring Framework verwendet. Das Spring-Ökosystem ist sehr gross. Es umfasst eine Vielzahl von Frameworks und Libraries, mit denen die wichtigsten Funktionalitäten für Applikationen im Business-Kontext, für Webapplikationen oder für API's umgesetzt werden können. Der Kern des Frameworks vollbringt als eine der wichtigsten Aufgaben die Dependency Injection, d.h. das Erstellen und die Verschaltung der Objekte, genannt „Beans“, d.h. der Komponenten der Applikation. Dies alles wurde in Kapitel 4 beschrieben. Abbildung 6.2. zeigt die wichtigsten Module des Spring Frameworks.

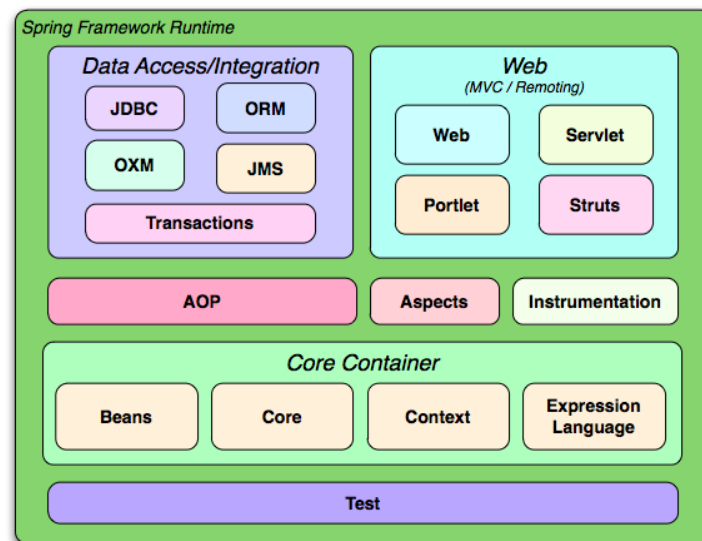


Abbildung 6.2.: Spring Framework Core

Die API wurde als Spring Boot-Applikation programmiert. Spring Boot bietet als wichtigste Zusatzkomponente die automatische Konfiguration der Applikation.

ORM

Objektrelaionales Mapping ORM bezeichnet die automatische Umwandlung des Domain-Models, d.h. der Klassen und Objekte aus der objektorientierten Programmiersprache in entsprechende Datenbanktabellen und -transaktionen. Diese Technologie wurde hier angewendet. Dazu mussten die Klassen mit Annotationen zur korrekten Konfiguration für die Umwandlung und Generierung

der Tabellen und Queries ergänzt werden. Dies war eine nicht triviale Aufgabe, auch wenn vom Umfang her sehr wenig Quellcode geschrieben werden musste. Die dazu verwendete Technologie war die Java Persistence API (JPA) mit der verbreitetsten Implementation Hibernate. Diese Technologien wurden ebenfalls in Kapitel 4 beschrieben.

6.1.5. Implementation der API

Mit der Spring-Initializr-homepage wurde das Projekt mit den wichtigsten Dependencies relativ schnell und einfach aufgesetzt, heruntergeladen und als Maven-Projekt in die IDE, Eclipse, geladen. Die wichtigsten Komponenten, die für die API zu programmieren waren, waren die Entity-Klassen, welche die Entitäten, die in der Datenbank abgebildet werden, repräsentieren, die Repository-Interfaces, mit welchen Zugriffe auf die Datenbank durchgeführt wird, sowie die Controller-Klassen, welche die HTTP-Requests auf die definierten Endpoints verarbeiten. Dazu wurden Datentemplate-Klassen erstellt für die Übergabe von Daten an die Controller sowie eine SwaggerConfig-Klasse für die Erstellung der Swagger-Dokumentation der API. Abbildung 6.3. gibt einen Überblick über die Klassen und Packages, die programmiert respektive erstellt wurden, sowie über die Ordnerstruktur, die von Spring Boot so vorgegeben ist. In Kapitel 5 wurden diese Komponenten beschrieben.

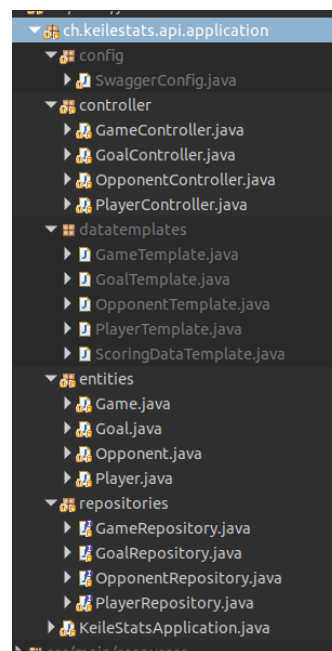


Abbildung 6.3.: Ordnerstruktur, Packages, Interfaces und Klassen

Listing 6.1. zeigt exemplarisch einen Ausschnitt aus der Klasse `GoalController`, der die Implementation eines Endpoints mit einem GET-Requests auf die Tor-Entität zeigt.

```
1 @GetMapping("/goals/{goalId}")
```

```
2 public ResponseEntity<Object> getGoalById(@PathVariable("goalId") Long goalId) {  
3  
4     Optional<Goal> optionalGoal = goalRepository.findById(goalId);  
5     if (optionalGoal.isPresent()) {  
6         return new ResponseEntity<>(optionalGoal.get(), HttpStatus.OK);  
7     } else {  
8         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("goal not found");  
9     }  
10 }
```

Listing 6.1: snippet Endpoint GET-Request auf Tor-Entität

Im Verlaufe der Programmierung wurde die API ständig mit Swagger getestet und entsprechend angepasst und verbessert. Dazu wurde von Anfang an parallel der vorliegende Bericht verfasst. Nach vielem Recherchieren, Programmieren, Testen, Beschreiben, Verbessern, Fehler beheben und all dem immer wieder von vorne liegt nun die API sowie dieser Bericht vor. Das Projekt ist auf Github unter <https://github.com/MarcRaemy/keilestats-app.git> verfügbar.

6.2. Probleme und Herausforderungen

6.2.1. Hoher Automatisierungsgrad von Spring

Mit Spring und Spring Boot kann man extrem schnell und effizient eine Applikation erstellen, die hohen Anforderungen genügt und komplexe und vielfältige Funktionen erfüllt - vorausgesetzt man weiss, was man macht und wie das Framework im Hintergrund funktioniert. Wenn man das erste Mal mit Datenbankzugriffen in Java, mit Annotationen, mit extern konfigurierten Applikationen, mit ORM und mit REST-Endpoints zu tun hat, ist das Programmieren mit Spring sehr anspruchsvoll, weil man nicht sieht, was unsichtbar im Hintergrund alles geschieht und die angewendeten Konzepte nicht kennt und versteht. Somit versteht man nicht, wie die Applikation überhaupt funktioniert und weiss nicht, wie man das Geschehen kontrollieren kann.

Für diese Arbeit mussten an diesem Punkt die Grundlagenbücher zu Spring, Hibernate und REST durchgearbeitet werden, um zumindest ein wenig eine Ahnung zu haben, wie Spring und ORM im Hintergrund funktionieren und was in der Applikation, die man programmiert, ungefähr geschieht. Dies bleibt jedoch mit Spring, bei geringem Erfahrungslevel, eine permanente Herausforderung.

6.2.2. Konfiguration des objektrelationalen Mappings

Damit das objektrelationale Mapping einwandfrei funktioniert und in der Datenbank tatsächlich das geschieht, was man will, müssen die richtigen Annotationen mit den richtigen Attributen am richtigen Ort gesetzt werden. Auch dies scheint Anfangs eine eher kleine Aufgabe zu sein (eine überschaubare Anzahl Klassen mit einer überschaubaren Anzahl Annotationen ergänzen). Um dies aber richtig zu machen und kontrollieren zu können, was passiert, müssen ebenfalls zuerst die Grundlagen erarbeitet werden zu Hibernate, Datenbanktransaktionen und annotationsbasierter Konfiguration. Das ist eine nicht zu unterschätzende Arbeit. Erschwerend kommt hinzu, dass die Onlinereisourcen und -Dokumentationen zwar vieles abdecken, jedoch sehr wenig fundiert erklären. Auch hier war deshalb zuerst das Studium von Grundlagenliteratur und sehr viel Ausprobieren nötig, um das nötige Verständnis zu erreichen und schliesslich die Annotationen korrekt und mit einem gewissen Verständnis dafür, was man tut, zu platzieren.

6.2.3. Breites Themengebiet

Hibernate und die Java Persistence API (JPA) sind grosse Frameworks, die komplexe Arbeiten verrichten, schon nur diese fundiert zu kennen und zu verstehen, bietet Stoff für eine oder sogar viele Arbeiten (beispielsweise zum Thema Optimierung der SQL-Generierung von Hibernate oder Ähnlichem). Zweitens ist der Bereich REST, Webservices und HTTP ebenfalls ein breites Themengebiet, in dem man sehr viele Aspekte einzelnen Arbeiten vertieft bearbeiten könnte. Schliesslich umfassen auch Spring und Spring Boot unzählige Funktionalitäten und Aspekte, die man in einzelnen Arbeiten im Detail untersuchen könnte.

Die vorliegende Arbeit umfasst also ein breites Themengebiet. Dadurch konnten nicht alle Konzepte und Technologien in grosser Tiefe und in grossem Umfang analysiert und bearbeitet werden. Es musste, sobald das für diese Arbeit nötige Wissen erreicht war, ein Punkt gemacht werden, da sonst der Umfang der Arbeit zu gross geworden wäre.

Dennoch war auch in diesem Kontext eine mehr oder weniger vertiefte Auseinandersetzung mit den verschiedenen Themen nötig, um präzise und korrekte Beschreibungen machen zu können. Zudem war das Ziel der Arbeit nicht, jedes Thema bis ins Detail zu durchleuchten, sondern den praxisbezogenen Anwendungsfall des HC Keile zu bearbeiten und praktische Erfahrung mit Java und Spring zu sammeln. Dieses Ziel wurde trotz allen Hindernissen gut erreicht.

A

Abkürzungen

ANSI	American National Standards Institute
AOP	Aspect Oriented Programming
API	Application Programming Interface
BPMN2	Business Process Model and Notation
CSS	Cascading Style Sheet
DBMS	Database Management System
DI	Dependency Injection
DOM	Document Object Model
EJB	Enterprise Java Beans
ERM	Entity Relationship Model
HATEOAS	Hypermedia as the Engine of Application State
HL7	Health Level 7
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
JDBC	Java Database Connectivity
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
JSP	Java Server Pages
MIME	Multipurpose Internet Mail Extensions
MVC	Model-View-Controller
ORM	Objekt/Relationales Mapping
PHP	Hypertext Processor
POJO	Plain Old Java Objects
POM	Project Object Model
REST	Representational State Transfer
RMI	Remote Method Invocation
ROA	Resource Oriented Architecture
SAX	Simple API for XML
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
URI	Unified Resource Identifier

URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Description Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

B

Quellcode und Dokumente

Im Folgenden ist der vollständige Quellcode des Projekts aufgelistet:

B.1. Main-Klasse

```
1 package ch.keilestats.app;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.CommandLineRunner;
10 import org.springframework.boot.SpringApplication;
11 import org.springframework.boot.autoconfigure.SpringBootApplication;
12
13 import ch.keilestats.app.entities.Game;
14 import ch.keilestats.app.entities.Goal;
15 import ch.keilestats.app.entities.Opponent;
16 import ch.keilestats.app.entities.Player;
17 import ch.keilestats.app.repositories.GameRepository;
18 import ch.keilestats.app.repositories.GoalRepository;
19 import ch.keilestats.app.repositories.OpponentRepository;
20 import ch.keilestats.app.repositories.PlayerRepository;
21
22 /* Annotation @SpringBootApplication: Indicates a configuration class that
23  * declares one or more @Bean methods and also triggers auto-configuration,
24  * component scanning, and configuration properties scanning. This is a convenience
25  * annotation that is equivalent to declaring @Configuration,
26  * @EnableAutoConfiguration, @ComponentScan.
27  */
28 @SpringBootApplication
29 /*
30  * Class SpringApplication bootstraps the Application: -creates an
31  * ApplicationContext instance -registers a CommandLinePropertySource to expose
32  * command line arguments as spring properties -refresh application context,
33  * loading all singleton beans -Trigger Any command line runner beans
34  */
35 public class KeileStatsApplication implements CommandLineRunner {
```

```
36
37     Logger logger = LoggerFactory.getLogger(getClass());
38
39     // Repositories used for setUpData()-Method in order to create sample-Data to test
40     // the API
41     @Autowired
42     GameRepository gameRepository;
43
44     @Autowired
45     PlayerRepository playerRepository;
46
47     @Autowired
48     OpponentRepository opponentRepository;
49
50     @Autowired
51     GoalRepository goalRepository;
52
53     public static void main(String[] args) {
54
55         SpringApplication.run(KeileStatsApplication.class, args);
56         System.out.println("Hello");
57     }
58
59     @Override
60     public void run(String... args) throws Exception {
61         // TODO Auto-generated method stub
62         setUpData();
63     }
64
65     /* Create some Data for testing purpose and save in Database */
66     public void setUpData() {
67
68         // create and 3 Players
69         Player player1 = new Player("Mueller", "Max");
70         Player player2 = new Player("Meier", "Erich");
71         Player player3 = new Player("Hugentobler", "Daniel");
72
73         // Create Set of Players that played each game (both the same here)
74         List<Player> playerKeileGame1 = new ArrayList<>();
75
76         playerKeileGame1.add(player1);
77         playerKeileGame1.add(player2);
78         playerKeileGame1.add(player3);
79
80         List<Player> playerKeileGame2 = new ArrayList<>();
81
82         playerKeileGame2.add(player1);
83         playerKeileGame2.add(player2);
84         playerKeileGame2.add(player3);
85
86         // save players to the database
87         playerRepository.save(player1);
88         playerRepository.save(player2);
89         playerRepository.save(player3);
90
91         // Create 2 Opponents
92         Opponent opponent1 = new Opponent("HC Gurmels Senioren");
93         Opponent opponent2 = new Opponent("HC Tiletz");
```

```
93 // Save Opponents to the database
94 opponentRepository.save(opponent1);
95 opponentRepository.save(opponent2);
96
97 // Create two games
98 Game game1 = new Game();
99 Game game2 = new Game();
100
101 gameRepository.save(game1);
102 gameRepository.save(game2);
103
104 // Create Goals
105 Goal goal1 = new Goal(player1, player2, player3, game1);
106 Goal goal2 = new Goal(player2, player1, game1);
107 Goal goal3 = new Goal(player1, game1);
108 Goal goal4 = new Goal(player3, player1, player2, game2);
109 Goal goal5 = new Goal(player2, player3, player1, game2);
110 Goal goal6 = new Goal(player1, game2);
111 Goal goal7 = new Goal(player1, player2, player3, game2);
112
113 // Assign Goals to 2 different Games
114 List<Goal> goalsKeileGame1 = new ArrayList<>();
115
116 goalsKeileGame1.add(goal1);
117 goalsKeileGame1.add(goal2);
118 goalsKeileGame1.add(goal3);
119
120 List<Goal> goalsKeileGame2 = new ArrayList<>();
121
122 goalsKeileGame2.add(goal4);
123 goalsKeileGame2.add(goal5);
124 goalsKeileGame2.add(goal6);
125 goalsKeileGame2.add(goal7);
126
127 // Save Goals to the database
128 goalRepository.save(goal1);
129 goalRepository.save(goal2);
130 goalRepository.save(goal3);
131 goalRepository.save(goal4);
132 goalRepository.save(goal5);
133 goalRepository.save(goal6);
134 goalRepository.save(goal7);
135
136 // Create 2 Games
137 game1.setGameDate("15.10.2017");
138 game1.setOpponent(opponent1);
139 game1.setGoalsKeile(goalsKeileGame1);
140 game1.setPlayers(playerKeileGame1);
141 game1.setNbGoalsOpponent(2);
142 game1.setNbGoalsKeile(goalsKeileGame1.size());
143
144 game2.setGameDate("7.11.2018");
145 game2.setOpponent(opponent2);
146 game2.setGoalsKeile(goalsKeileGame2);
147 game2.setPlayers(playerKeileGame2);
148 game2.setNbGoalsOpponent(1);
149 game2.setNbGoalsKeile(goalsKeileGame2.size());
150
```



```

151
152     // Save games to database
153     gameRepository.save(game1);
154     gameRepository.save(game2);
155
156 }
157 }

```

Listing B.1: Klasse Main vollständig

B.2. Package config

```

1 package ch.keilestats.app.config;
2
3
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6
7 import springfox.documentation.builders.RequestHandlerSelectors;
8 import springfox.documentation.service.ApiInfo;
9 import springfox.documentation.spi.DocumentationType;
10 import springfox.documentation.spring.web.plugins.Docket;
11 import springfox.documentation.swagger2.annotations.EnableSwagger2;
12
13 import static springfox.documentation.builders.PathSelectors.regex;
14
15 @EnableSwagger2
16 /*Indicates that Swagger support should be enabled.
17  * This should have an accompanying '@Configuration' annotation.
18  * Loads all required beans.
19  * */
20 @Configuration
21 /*Indicates that a class declares one or more @Bean methods and may be
22 processed by the Spring container to generate bean definitions
23 and service requests for those beans at runtime*/
24 public class SwaggerConfig {
25
26     /* Returns the Swagger2 Interface */
27     @Bean
28     public Docket keileAPI() {
29         return new Docket(DocumentationType.SWAGGER_2).select()
30             .apis(RequestHandlerSelectors.basePackage("ch.keilestats.app")).paths(
31                 regex("/app.*"))
32             .build().apiInfo(metaInfo());
33     }
34
35     // Titletext to be displayed in the Swagger interface
36     private ApiInfo metaInfo() {
37
38         ApiInfo apiInfo = new ApiInfo("Keile Stats API",
39             "API for managing " + "statistics of a just-for-fun Icehockey Team", "
40             1.0", "Terms of Service", "", "",
41             "");
42
43         return apiInfo;
44     }
45 }

```

```
42 }  
43 }
```

Listing B.2: Klasse SwaggerConfig vollständig

B.3. Package datatemplates

```
1 package ch.keilestats.app.datatemplates;  
2  
3  
4 import java.util.Arrays;  
5 import java.util.List;  
6  
7 /*Class to help to pass data to the POST- and PUT-methods of the game controller.  
8  * Template for collecting, saving and presenting game data*/  
9 public class GameTemplate {  
10  
11     String gameDate;  
12     String opponentName;  
13     Long[] playerIdList;  
14     List<GoalTemplate> goalTemplateList;  
15     Integer nbGoalsOpponent;  
16     Integer nbGoalsKeile;  
17  
18     public GameTemplate() {  
19  
20     }  
21  
22     public GameTemplate(String gameDate, String opponent, Long[] playersList, List<  
23         GoalTemplate> goalsList,  
24         Integer nbGoalsOpponent, Integer nbGoalsKeile) {  
25         super();  
26         this.gameDate = gameDate;  
27         this.opponentName = opponent;  
28         this.playerIdList = playersList;  
29         this.goalTemplateList = goalsList;  
30         this.nbGoalsOpponent = nbGoalsOpponent;  
31         this.nbGoalsKeile = nbGoalsKeile;  
32     }  
33  
34     public String getGameDate() {  
35         return gameDate;  
36     }  
37  
38     public void setGameDate(String gameDate) {  
39         this.gameDate = gameDate;  
40     }  
41  
42     public Long[] getPlayerIdList() {  
43         return playerIdList;  
44     }  
45  
46     public void setPlayerIdList(Long[] playersList) {  
47         this.playerIdList = playersList;  
48     }  
49 }
```

```

48
49     public List<GoalTemplate> getGoalsList() {
50         return goalTemplateList;
51     }
52
53     public void setGoalsList(List<GoalTemplate> goalsList) {
54         this.goalTemplateList = goalsList;
55     }
56
57     public Integer getNbGoalsOpponent() {
58         return nbGoalsOpponent;
59     }
60
61     public void setNbGoalsOpponent(Integer nbGoalsOpponent) {
62         this.nbGoalsOpponent = nbGoalsOpponent;
63     }
64
65     public String getOpponentName() {
66         return opponentName;
67     }
68
69     public void setOpponentName(String opponentName) {
70         this.opponentName = opponentName;
71     }
72
73     public Integer getNbGoalsKeile() {
74         return nbGoalsKeile;
75     }
76
77     public void setNbGoalsKeile(Integer nbGoalsKeile) {
78         this.nbGoalsKeile = nbGoalsKeile;
79     }
80
81     @Override
82     public String toString() {
83         return "GameTemplate [gameDate=" + gameDate + ", opponentName=" + opponentName
84             + ", playerIdList="
85             + Arrays.toString(playerIdList) + ", goalTemplateList=" +
86             goalTemplateList + ", nbGoalsOpponent="
87             + nbGoalsOpponent + ", nbGoalsKeile=" + nbGoalsKeile + "];"
88     }

```

Listing B.3: Klasse GameTemplate vollständig

```

1 package ch.keilestats.app.datatemplates;
2
3 /*
4  * Class to help to pass data to the POST and PUT-methods of the
5  * goal controller. Template for collecting, saving and
6  * presenting goal data*
7  */
8 public class GoalTemplate {
9
10     Long goalScorerId;

```

```

11     Long firstAssistantId;
12     Long secondAssistantId;
13
14     public GoalTemplate() {
15     }
16
17     public Long getGoalScorerId() {
18         return goalScorerId;
19     }
20
21     public void setGoalScorerId(Long goalScorerId) {
22         this.goalScorerId = goalScorerId;
23     }
24
25     public Long getFirstAssistantId() {
26         return firstAssistantId;
27     }
28
29     public void setFirstAssistantId(Long firstAssistantId) {
30         this.firstAssistantId = firstAssistantId;
31     }
32
33     public Long getSecondAssistantId() {
34         return secondAssistantId;
35     }
36
37     public void setSecondAssistantId(Long secondAssistantId) {
38         this.secondAssistantId = secondAssistantId;
39     }
40
41     @Override
42     public String toString() {
43         return "GoalTemplate [goalScorerId=" + goalScorerId + ", firstAssistantId=" +
44             firstAssistantId
45             + ", secondAssistangId=" + secondAssistantId + "]";
46     }
47 }

```

Listing B.4: Klasse GoalTemplate vollständig

```

1 package ch.keilestats.app.datatemplates;
2
3
4 /*Class to help to pass data to the POST- and PUT method of the opponent controller.
5  * Template for collecting, saving and presenting Opponent data*/
6 public class OpponentTemplate {
7
8     String opponentName;
9
10    public OpponentTemplate() {
11    }
12
13    public OpponentTemplate(String opponentName) {
14        super();
15        this.opponentName = opponentName;

```

```
16     }
17
18     public String getOpponentName() {
19         return opponentName;
20     }
21
22     public void setOpponentName(String opponentName) {
23         this.opponentName = opponentName;
24     }
25
26     @Override
27     public String toString() {
28         return "OpponentTemplate [opponentName=" + opponentName + "]";
29     }
30 }
```

Listing B.5: Klasse OpponentTemplate vollständig

```
1 package ch.keilestats.app.datatemplates;
2
3 /*Class to help to pass data to the POST- and PUT method of the player controller.
4  * Template for collecting, saving and presenting player data*/
5 public class PlayerTemplate {
6
7     private String lastname;
8     private String firstname;
9     private String position;
10    private String email;
11    private String address;
12    private String phone;
13
14    public PlayerTemplate() {
15    }
16
17    public PlayerTemplate(String lastname, String firstname, String position, String
18        email, String address,
19        String phone) {
20        super();
21        this.lastname = lastname;
22        this.firstname = firstname;
23        this.position = position;
24        this.email = email;
25        this.address = address;
26        this.phone = phone;
27    }
28
29    public String getLastname() {
30        return lastname;
31    }
32
33    public void setLastname(String lastname) {
34        this.lastname = lastname;
35    }
36
37    public String getFirstname() {
38        return firstname;
39    }
39 }
```

```
38     }
39
40     public void setFirstname(String firstname) {
41         this.firstname = firstname;
42     }
43
44     public String getPosition() {
45         return position;
46     }
47
48     public void setPosition(String position) {
49         this.position = position;
50     }
51
52     public String getEmail() {
53         return email;
54     }
55
56     public void setEmail(String email) {
57         this.email = email;
58     }
59
60     public String getAddress() {
61         return address;
62     }
63
64     public void setAddress(String address) {
65         this.address = address;
66     }
67
68     public String getPhone() {
69         return phone;
70     }
71
72     public void setPhone(String phone) {
73         this.phone = phone;
74     }
75
76     @Override
77     public int hashCode() {
78         final int prime = 31;
79         int result = 1;
80         result = prime * result + ((firstname == null) ? 0 : firstname.hashCode());
81         result = prime * result + ((lastname == null) ? 0 : lastname.hashCode());
82         return result;
83     }
84
85     //only considers firstname and lastname
86     @Override
87     public boolean equals(Object obj) {
88         if (this == obj)
89             return true;
90         if (obj == null)
91             return false;
92         if (getClass() != obj.getClass())
93             return false;
94         PlayerTemplate other = (PlayerTemplate) obj;
95         if (firstname == null) {
```

```

96         if (other.firstname != null)
97             return false;
98     } else if (!firstname.equals(other.firstname))
99         return false;
100     if (lastname == null) {
101         if (other.lastname != null)
102             return false;
103     } else if (!lastname.equals(other.lastname))
104         return false;
105     return true;
106 }
107
108 @Override
109 public String toString() {
110     return "PlayerTemplate [lastname=" + lastname + ", firstname=" + firstname + ",
111         position=" + position
112         + ", email=" + email + ", address=" + address + ", phone=" + phone + "]"
113     ;
114 }
115 }

```

Listing B.6: Klasse PlayerTemplate vollständig

```

1 package ch.keilestats.app.datatemplates;
2
3
4 /*Class to help presenting scoring data overview for the endpoint GET players/
   scoringtable*/
5 public class ScoringDataTemplate {
6
7     Long playerId;
8     String playerFirstName;
9     String playerLastName;
10    Integer assistsScored;
11    Integer goalsScored;
12    Integer totalPoints;
13    Integer gamesPlayed;
14
15    public ScoringDataTemplate() {
16    }
17
18    public ScoringDataTemplate(Long playerId, String playerFirstName, String
19        playerLastName, Integer assistsScored,
20        Integer goalsScored, Integer totalPoints, Integer gamesPlayed) {
21        this.playerId = playerId;
22        this.playerFirstName = playerFirstName;
23        this.playerLastName = playerLastName;
24        this.assistsScored = assistsScored;
25        this.goalsScored = goalsScored;
26        this.totalPoints = totalPoints;
27        this.gamesPlayed = gamesPlayed;
28    }
29
30    public Long getPlayerId() {
31        return playerId;
32    }
33 }

```

```
32
33     public void setPlayerId(Long playerId) {
34         this.playerId = playerId;
35     }
36
37     public String getPlayerFirstName() {
38         return playerFirstName;
39     }
40
41     public void setPlayerFirstName(String playerFirstName) {
42         this.playerFirstName = playerFirstName;
43     }
44
45     public String getPlayerLastName() {
46         return playerLastName;
47     }
48
49     public void setPlayerLastName(String playerLastName) {
50         this.playerLastName = playerLastName;
51     }
52
53     public Integer getAssistsScored() {
54         return assistsScored;
55     }
56
57     public void setAssistsScored(Integer assistsScored) {
58         this.assistsScored = assistsScored;
59     }
60
61     public Integer getGoalsScored() {
62         return goalsScored;
63     }
64
65     public void setGoalsScored(Integer goalsScored) {
66         this.goalsScored = goalsScored;
67     }
68
69     public Integer getTotalPoints() {
70         return totalPoints;
71     }
72
73     public void setTotalPoints(Integer totalPoints) {
74         this.totalPoints = totalPoints;
75     }
76
77     public Integer getGamesPlayed() {
78         return gamesPlayed;
79     }
80
81     public void setGamesPlayed(Integer gamesPlayed) {
82         this.gamesPlayed = gamesPlayed;
83     }
84
85     @Override
86     public String toString() {
87         return "ScoringDataTemplate [playerId=" + playerId + ", playerFirstName=" +
            playerFirstName
```



```

88         + ", playerLastName=" + playerLastName + ", assistsScored=" +
            assistsScored + ", goalsScored="
89         + goalsScored + ", totalPoints=" + totalPoints + ", gamesPlayed=" +
            gamesPlayed + "];
90     }
91 }

```

Listing B.7: Klasse ScoringDataTemplate vollständig

B.4. Package entities

```

1 package ch.keilestats.app.entities;
2
3
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import javax.persistence.CascadeType;
8 import javax.persistence.Column;
9 import javax.persistence.Entity;
10 import javax.persistence.GeneratedValue;
11 import javax.persistence.GenerationType;
12 import javax.persistence.Id;
13 import javax.persistence.JoinColumn;
14 import javax.persistence.JoinTable;
15 import javax.persistence.ManyToMany;
16 import javax.persistence.ManyToOne;
17 import javax.persistence.OneToOne;
18
19 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
20 import com.fasterxml.jackson.annotation.JsonManagedReference;
21
22 @Entity
23 /*
24  * @Entity = Annotation that marks the Class to JPA as a persistent Entity. And
25  * indicates to the framework as a bean to instantiate for the application.
26  */
27 public class Game {
28
29     @Id
30     @GeneratedValue(strategy = GenerationType.IDENTITY)
31     @Column(nullable = false)
32     private Long gameId;
33
34     private String gameDate;
35
36     // Cascade attribute defines how changes on one side of the association are
37     // "cascaded" to the other side.
38     @OneToMany(mappedBy = "game", cascade = CascadeType.REMOVE)
39     private List<Goal> goalsKeile = new ArrayList<>();
40
41     private Integer nbGoalsKeile;
42     private Integer nbGoalsOpponent;
43
44     @ManyToOne

```

```

45 @JoinColumn(name = "OPPONENT_ID") // marks and names column, where the foreign key
    of the related entity is saved
46 private Opponent opponent;
47
48 @ManyToMany
49 // Definition of the columns of the new table emerging from the many-to-many
50 // relationship
51 @JoinTable(joinColumns = @JoinColumn(name = "game_id"), inverseJoinColumns =
    @JoinColumn(name = "player_id"))
52 @JsonIgnoreProperties("games")
53 private List<Player> players = new ArrayList<>();
54
55 public Game() {
56 }
57
58 public Game(String gameDate, Opponent opponent, Integer nbGoalsKeile, Integer
    goalsOpponent, List<Player> players,
59     List<Goal> goalsKeile) {
60
61     this.gameDate = gameDate;
62     this.goalsKeile = goalsKeile;
63     this.nbGoalsKeile = nbGoalsKeile;
64     this.nbGoalsOpponent = goalsOpponent;
65     this.opponent = opponent;
66     this.players = players;
67 }
68
69 public Long getGameId() {
70     return gameId;
71 }
72
73 public void setGameId(Long id) {
74     this.gameId = id;
75 }
76
77 public String getGameDate() {
78     return gameDate;
79 }
80
81 public void setGameDate(String date) {
82
83     this.gameDate = date;
84 }
85
86 @JsonManagedReference(value = "GoalinGame")
87 /*
88  * Annotation used to indicate that annotated property is part of two-way
89  * linkage between fields; and that its role is "parent" (or "forward") link.
90  * Necessary to break infinite loop at serialisation and deserialisation
91  */
92 public List<Goal> getGoalsKeile() {
93     return goalsKeile;
94 }
95
96 public void setGoalsKeile(List<Goal> goals) {
97
98     this.goalsKeile = goals;
99 }

```

```

100
101     public int getNbGoalsOpponent() {
102         return nbGoalsOpponent;
103     }
104
105     public void setNbGoalsOpponent(Integer nbGoalsOpponent) {
106         this.nbGoalsOpponent = nbGoalsOpponent;
107     }
108
109     @JsonManagedReference(value = "opponentInGame")
110     public Opponent getOpponent() {
111         return opponent;
112     }
113
114     public void setOpponent(Opponent opponent) {
115         this.opponent = opponent;
116     }
117
118     @JsonManagedReference
119     public List<Player> getPlayers() {
120         return players;
121     }
122
123     public void setPlayers(List<Player> players) {
124
125         this.players = players;
126     }
127
128     public Integer getNbGoalsKeile() {
129         return nbGoalsKeile;
130     }
131
132     public void setNbGoalsKeile(Integer nbGoalsKeile) {
133         this.nbGoalsKeile = nbGoalsKeile;
134     }
135
136     @Override
137     public int hashCode() {
138         final int prime = 31;
139         int result = 1;
140         result = prime * result + ((gameDate == null) ? 0 : gameDate.hashCode());
141         result = prime * result + ((goalsKeile == null) ? 0 : goalsKeile.hashCode());
142         result = prime * result + ((nbGoalsKeile == null) ? 0 : nbGoalsKeile.hashCode());
143         result = prime * result + ((nbGoalsOpponent == null) ? 0 : nbGoalsOpponent.
144             hashCode());
145         result = prime * result + ((opponent == null) ? 0 : opponent.hashCode());
146         return result;
147     }
148
149     @Override
150     public boolean equals(Object obj) {
151         if (this == obj)
152             return true;
153         if (obj == null)
154             return false;
155         if (getClass() != obj.getClass())
156             return false;

```

```

156     Game other = (Game) obj;
157     if (gameDate == null) {
158         if (other.gameDate != null)
159             return false;
160     } else if (!gameDate.equals(other.gameDate))
161         return false;
162     if (goalsKeile == null) {
163         if (other.goalsKeile != null)
164             return false;
165     } else if (!goalsKeile.equals(other.goalsKeile))
166         return false;
167     if (nbGoalsKeile == null) {
168         if (other.nbGoalsKeile != null)
169             return false;
170     } else if (!nbGoalsKeile.equals(other.nbGoalsKeile))
171         return false;
172     if (nbGoalsOpponent == null) {
173         if (other.nbGoalsOpponent != null)
174             return false;
175     } else if (!nbGoalsOpponent.equals(other.nbGoalsOpponent))
176         return false;
177     if (opponent == null) {
178         if (other.opponent != null)
179             return false;
180     } else if (!opponent.equals(other.opponent))
181         return false;
182     return true;
183 }
184
185 @Override
186 public String toString() {
187
188     return "Game [gameId = " + gameId + ", gameDate = " + gameDate + ", goals_keile
189         = " + goalsKeile
190         + ", goals_opponent=" + nbGoalsOpponent + ", opponent=" + opponent + ",
191         players=" + players + "];"

```

Listing B.8: Klasse Game vollständig

```

1 package ch.keilestats.app.entities;
2
3
4 import javax.persistence.Column;
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.GenerationType;
8 import javax.persistence.Id;
9 import javax.persistence.JoinColumn;
10 import javax.persistence.ManyToOne;
11
12
13 import com.fasterxml.jackson.annotation.JsonBackReference;
14 import com.fasterxml.jackson.annotation.JsonManagedReference;
15

```

```
16 @Entity /*
17     * Annotation that marks the Class to JPA as a persistent Entity. And indicates
18     * to the framework as a bean to instantiate for the application.
19     */
20 public class Goal {
21
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     @Column(nullable = false)
25     private Long goalId;
26
27     @ManyToOne
28     @JoinColumn(name = "GAME_ID") // marks and names column, where the foreign key of
        the related entity is saved
29     private Game game;
30
31     @ManyToOne
32     @JoinColumn(name = "SCORER_ID")
33     private Player goalScorer;
34
35     @ManyToOne
36     @JoinColumn(name = "ASSISTANT1_ID")
37     private Player firstAssistant;
38
39     @ManyToOne
40     @JoinColumn(name = "ASSISTANT2_ID")
41     private Player secondAssistant;
42
43     public Goal() {}; // Empty constructor required by the framework
44
45     public Goal(Player goalScorer, Game game) {
46
47         this.goalScorer = goalScorer;
48         this.game = game;
49     }
50
51     public Goal(Player goalScorer, Player firstAssistant, Game game) {
52
53         this.goalScorer = goalScorer;
54         this.firstAssistant = firstAssistant;
55         this.game = game;
56     }
57
58     public Goal(Player goalScorer, Player firstAssistant, Player secondAssistant, Game
        game) {
59
60         this.goalScorer = goalScorer;
61         this.firstAssistant = firstAssistant;
62         this.secondAssistant = secondAssistant;
63         this.game = game;
64     }
65
66     public Long getGoalId() {
67         return goalId;
68     }
69
70     public void setGoalId(Long id) {
71         this.goalId = id;
```

```

72     }
73
74     @JsonBackReference(value = "gameInGoal")
75     /*
76      * Annotation used to indicate that associated property is part of two-way
77      * linkage between fields; and that its role is "child" (or "back") link. Used
78      * to prevent infinite loop at serialisation and deserialisation
79      */
80     public Game getGame() {
81         return game;
82     }
83
84     public void setGame(Game game) {
85         this.game = game;
86     }
87
88     @JsonManagedReference(value = "goalScorerInGoal")
89     /*
90      * Annotation used to indicate that annotated property is part of two-way
91      * linkage between fields; and that its role is "parent" (or "forward") link.
92      * Necessary to break infinite loop at serialisation and deserialisation
93      */
94     public Player getGoalScorer() {
95         return goalScorer;
96     }
97
98     public void setGoalScorer(Player goalScorer) {
99         this.goalScorer = goalScorer;
100     }
101
102
103     @JsonManagedReference(value = "firstAssistantInGoal")
104     public Player getFirstAssistant() {
105         return firstAssistant;
106     }
107
108     public void setFirstAssistant(Player firstAssistant) {
109         this.firstAssistant = firstAssistant;
110     }
111
112     @JsonManagedReference(value = "secondAssistantInGoal")
113     public Player getSecondAssistant() {
114         return secondAssistant;
115     }
116
117     public void setSecondAssistant(Player assistant2) {
118         this.secondAssistant = assistant2;
119     }
120
121
122     @Override
123     public String toString() {
124         return "Goal [goalId=" + goalId + ", game=" + game + ", scorer=" + goalScorer +
125             ", assist1=" + firstAssistant
126             + ", assist2=" + secondAssistant + "];"
127     }

```

128 }

Listing B.9: Klasse Goal vollständig

```
1 package ch.keilestats.app.entities;
2
3
4 import java.util.ArrayList;
5 import java.util.List;
6
7 import javax.persistence.Column;
8 import javax.persistence.Entity;
9 import javax.persistence.GeneratedValue;
10 import javax.persistence.GenerationType;
11 import javax.persistence.Id;
12 import javax.persistence.OneToMany;
13
14 import com.fasterxml.jackson.annotation.JsonBackReference;
15
16 @Entity
17 public class Opponent {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     @Column(nullable = false)
22     private Long opponentId;
23
24     private String opponentName;
25
26     @OneToMany(mappedBy = "opponent")
27     private List<Game> games = new ArrayList<>();
28
29     // Empty constructor needed by Spring Boot
30     public Opponent() {
31     }
32
33     public Opponent(String name) {
34
35         this.opponentName = name;
36     }
37
38     public Opponent(String name, List<Game> games) {
39
40         this.opponentName = name;
41         this.games = games;
42     }
43
44     public Long getOpponentId() {
45         return opponentId;
46     }
47
48     public void setOpponentId(Long id) {
49         this.opponentId = id;
50     }
51
52     public String getOpponentName() {
```

```

53     return opponentName;
54 }
55
56 public void setOpponentName(String name) {
57     this.opponentName = name;
58 }
59
60 @JsonBackReference(value = "opponentInGame")
61 public List<Game> getGames() {
62     return games;
63 }
64
65 public void setGames(List<Game> games) {
66     this.games = games;
67 }
68
69 @Override
70 public int hashCode() {
71     final int prime = 31;
72     int result = 1;
73     result = prime * result + ((opponentName == null) ? 0 : opponentName.hashCode());
74     return result;
75 }
76
77 @Override
78 public boolean equals(Object obj) {
79     if (this == obj)
80         return true;
81     if (obj == null)
82         return false;
83     if (getClass() != obj.getClass())
84         return false;
85     Opponent other = (Opponent) obj;
86     if (opponentName == null) {
87         if (other.opponentName != null)
88             return false;
89     } else if (!opponentName.equals(other.opponentName))
90         return false;
91     return true;
92 }
93
94 @Override
95 public String toString() {
96     return "Opponent [opponentId=" + opponentId + ", opponentName=" + opponentName
97         + "games=" + games + "]";
98 }

```

Listing B.10: Klasse Opponent vollständig

```

1 package ch.keilestats.app.entities;
2
3
4 import java.util.ArrayList;
5 import java.util.List;

```



```

6
7 import javax.persistence.Column;
8 import javax.persistence.Entity;
9 import javax.persistence.GeneratedValue;
10 import javax.persistence.GenerationType;
11 import javax.persistence.Id;
12 import javax.persistence.ManyToMany;
13 import javax.persistence.OneToOne;
14
15 import com.fasterxml.jackson.annotation.JsonBackReference;
16 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
17
18 @Entity /*
19      * Annotation that marks the Class to JPA as a persistent Entity and indicates
20      * it to the framework as a bean to instantiate for the application.
21      */
22 @JsonIgnoreProperties(value = { "goalsScored", "firstAssists", "secondAssists" })
23 /*
24      * Annotation used to avoid infinite loop at serialisation and deserialisation
25      */
26 public class Player {
27
28     @Id // marks the attribute primary key to the persistence layer
29     @GeneratedValue(strategy = GenerationType.IDENTITY) // strategy to generate the
        primary key
30     @Column(nullable = false)
31     private Long playerId;
32     private String lastname;
33     private String firstname;
34     private String position;
35     private String email;
36     private String address;
37     private String phone;
38
39     /*
40      * Indicates the relationship, many-to-many, between the entities player and
41      * game
42      */
43     @ManyToMany(mappedBy = "players")
44     /* Annotation to break the infinite loop occurring at serialisation */
45     @JsonIgnoreProperties("players")
46     private List<Game> games = new ArrayList<>();
47
48     @OneToOne(mappedBy = "goalScorer")
49     /*
50      * Indicates the relation between the entity goal and player and that the
51      * relationship is mapped by the attribute "goalScorer" of the "Goal"-table.
52      * foreign key for the goal scorer is saved in the "Goal"-Table
53      */
54     private List<Goal> goalsScored = new ArrayList<>();
55
56     @OneToOne(mappedBy = "firstAssistant")
57     private List<Goal> firstAssists = new ArrayList<>();
58
59     @OneToOne(mappedBy = "secondAssistant")
60     private List<Goal> secondAssists = new ArrayList<>();
61
62     // Empty constructor required by the framework

```

```
63 public Player() {
64 }
65
66 public Player(String lastname, String firstname) {
67     this.lastname = lastname;
68     this.firstname = firstname;
69 }
70
71 public Player(String lastname, String firstname, String position, String email,
72               String address, String phone) {
73     this.lastname = lastname;
74     this.firstname = firstname;
75     this.position = position;
76     this.email = email;
77     this.address = address;
78     this.phone = phone;
79 }
80
81 public void setPlayerId(Long playerId) {
82     this.playerId = playerId;
83 }
84
85 public Long getPlayerId() {
86     return playerId;
87 }
88
89 public String getLastName() {
90     return lastname;
91 }
92
93 public void setLastName(String lastname) {
94     this.lastname = lastname;
95 }
96
97 public String getFirstname() {
98     return firstname;
99 }
100
101 public void setFirstname(String firstname) {
102     this.firstname = firstname;
103 }
104
105 public String getPosition() {
106     return position;
107 }
108
109 public void setPosition(String position) {
110     this.position = position;
111 }
112
113 public String getPhone() {
114     return phone;
115 }
116
117 public void setPhone(String phone) {
118     this.phone = phone;
119 }
```

```
120     }
121
122     public String getEmail() {
123         return email;
124     }
125
126     public void setEmail(String email) {
127         this.email = email;
128     }
129
130     public String getAddress() {
131         return address;
132     }
133
134     public void setAddress(String address) {
135         this.address = address;
136     }
137
138     @JsonBackReference
139     /*
140      * Annotation used to indicate that associated property is part of two-way
141      * linkage between fields; and that its role is "child" (or "back") link. Used
142      * to prevent infinite loop at serialisation and deserialisation
143      */
144     public List<Game> getGames() {
145         return games;
146     }
147
148     public void setGames(List<Game> games) {
149         this.games = games;
150     }
151
152     @JsonBackReference(value = "goalScorerInGoal")
153     public List<Goal> getGoalsScored() {
154         return goalsScored;
155     }
156
157     public void setGoalsScored(List<Goal> goalsScored) {
158         this.goalsScored = goalsScored;
159     }
160
161     @JsonBackReference(value = "firstAssistantInGoal")
162     public List<Goal> getFirstAssists() {
163         return firstAssists;
164     }
165
166     public void setFirstAssists(List<Goal> firstAssists) {
167         this.firstAssists = firstAssists;
168     }
169
170     @JsonBackReference(value = "secondAssistantInGoal")
171     public List<Goal> getSecondAssists() {
172         return secondAssists;
173     }
174
175     public void setSecondAssists(List<Goal> secondAssists) {
176         this.secondAssists = secondAssists;
177     }
```

```

178
179 @Override
180 public int hashCode() {
181     final int prime = 31;
182     int result = 1;
183     result = prime * result + ((firstname == null) ? 0 : firstname.hashCode());
184     result = prime * result + ((lastname == null) ? 0 : lastname.hashCode());
185     return result;
186 }
187
188 @Override
189 public boolean equals(Object obj) {
190     if (this == obj)
191         return true;
192     if (obj == null)
193         return false;
194     if (getClass() != obj.getClass())
195         return false;
196     Player other = (Player) obj;
197     if (firstname == null) {
198         if (other.firstname != null)
199             return false;
200     } else if (!firstname.equals(other.firstname))
201         return false;
202     if (lastname == null) {
203         if (other.lastname != null)
204             return false;
205     } else if (!lastname.equals(other.lastname))
206         return false;
207     return true;
208 }
209
210 @Override
211 public String toString() {
212     return "Player [playerId=" + playerId + ", lastname=" + lastname + ", firstname"
213         + " + firstname + ", position="
214         + position + ", address=" + address + ", phone=" + phone + ", email=" +
215         email + ", games=" + games
216         + "Goals=" + goalsScored + "Assist1=" + firstAssists + "Assist2=" +
217         secondAssists + "];"

```

Listing B.11: Klasse Player vollständig

B.5. Package repositories

```

1 package ch.keilestats.app.repositories;
2
3
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 import ch.keilestats.app.entities.Game;

```

```
8
9 @Repository
10 public interface GameRepository extends JpaRepository<Game, Long> {
11
12 }
```

Listing B.12: Interface GameRepository vollständig

```
1 package ch.keilestats.app.repositories;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 import ch.keilestats.app.entities.Goal;
7
8 @Repository
9 public interface GoalRepository extends JpaRepository<Goal, Long> {
10
11 }
```

Listing B.13: Interface GoalRepository vollständig

```
1 package ch.keilestats.app.repositories;
2
3 import java.util.Optional;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6 import ch.keilestats.app.entities.*;
7
8 @Repository
9 public interface OpponentRepository extends JpaRepository<Opponent, Long> {
10
11     Optional<Opponent> findByOpponentName(String opponentName);
12
13 }
```

Listing B.14: Interface OpponentRepository vollständig

```
1 package ch.keilestats.app.repositories;
2
3
4 import java.util.Optional;
5
6 import org.springframework.data.jpa.repository.JpaRepository;
7 import org.springframework.stereotype.Repository;
8
9 import ch.keilestats.app.entities.*;
10
11 @Repository
12 public interface PlayerRepository extends JpaRepository<Player, Long> {
13
14     Optional<Player> findByFirstnameAndLastname(String firstname, String lastname);
15 }
```

16 }

Listing B.15: Interface PlayerRepository vollständig

B.6. Package controller

```

1 package ch.keilestats.app.controller;
2
3 import java.net.URI;
4 import java.util.ArrayList;
5 import java.util.List;
6 import java.util.Optional;
7
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.http.MediaType;
11 import org.springframework.http.ResponseEntity;
12 import org.springframework.web.bind.annotation.DeleteMapping;
13 import org.springframework.web.bind.annotation.GetMapping;
14 import org.springframework.web.bind.annotation.PathVariable;
15 import org.springframework.web.bind.annotation.PostMapping;
16 import org.springframework.web.bind.annotation.PutMapping;
17 import org.springframework.web.bind.annotation.RequestBody;
18 import org.springframework.web.bind.annotation.RequestMapping;
19 import org.springframework.web.bind.annotation.RestController;
20 import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
21
22 import ch.keilestats.app.datatemplates.GameTemplate;
23 import ch.keilestats.app.datatemplates.GoalTemplate;
24 import ch.keilestats.app.entities.Game;
25 import ch.keilestats.app.entities.Goal;
26 import ch.keilestats.app.entities.Opponent;
27 import ch.keilestats.app.entities.Player;
28 import ch.keilestats.app.repositories.GameRepository;
29 import ch.keilestats.app.repositories.GoalRepository;
30 import ch.keilestats.app.repositories.OpponentRepository;
31 import ch.keilestats.app.repositories.PlayerRepository;
32
33 @RestController
34 /*
35  * A convenience annotation that is itself annotated with @Controller
36  * and @ResponseBody. @Controller is a specification of @Component to indicate
37  * to the framework that the container should instantiate the class as a bean
38  */
39 @RequestMapping("/app")
40 /*
41  * Annotation for mapping web requests onto methods in request-handling classes
42  * with flexible method signatures
43  */
44 public class GameController { // handles calls on game resources
45
46     @Autowired /*
47         * Marks a constructor, field, setter method, or config method as to be
48         * autowired by Spring's dependency injection facilities.
49         */

```

```

50 private GameRepository gameRepository;
51 @Autowired
52 private PlayerRepository playerRepository;
53 @Autowired
54 private OpponentRepository opponentRepository;
55 @Autowired
56 private GoalRepository goalRepository;
57
58 // Annotation for mapping HTTP GET requests onto specific handler methods.
59 @GetMapping("/games") // URI on which the get request is called.
60 public ResponseEntity<Object> getAllGames() {
61
62     return ResponseEntity.status(HttpStatus.OK).body(gameRepository.findAll());
63 }
64
65 // "ResponseEntity" used to return HTTP-StatusCodes and custom messages in the
66 // response body
67 @GetMapping("/games/{gameId}")
68 public ResponseEntity<Object> getGameById(@PathVariable("gameId") Long gameId) {
69
70     // check if game with id gameId is present in the database and return it or
71     // return error message
72     Optional<Game> optionalGame = gameRepository.findById(gameId);
73     if (optionalGame.isPresent()) {
74         return new ResponseEntity<>(optionalGame.get(), HttpStatus.OK);
75     }
76     return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("No game with id " +
77         gameId + " found");
78 }
79
80 @DeleteMapping("/games/{gameId}")
81 public ResponseEntity<Object> deleteGame(@PathVariable("gameId") Long gameId) {
82
83     if (gameRepository.findById(gameId).isPresent()) {
84         gameRepository.deleteById(gameId);
85         return ResponseEntity.status(HttpStatus.OK).body("game with id " + gameId +
86             " deleted");
87     }
88     else {
89         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("No game with id
90             " + gameId + " found");
91     }
92 }
93
94 /*
95  * Returning a ResponseEntity with a header containing the URL of the created
96  * resource. @RequestBody indicates that the Body of the Request should be bound
97  * to the Goal object, not some Header Parameters
98  */
99 @PostMapping(path = "/games", consumes = { MediaType.APPLICATION_JSON_VALUE, "
100     application/json" })
101 public ResponseEntity<Object> addGame(@RequestBody GameTemplate gameTemplate) {
102
103     Game game = new Game();
104     gameRepository.save(game);

```

```

104 // Handle Opponent: if Opponent with the given Name present in Database, take
105 // existing. Else create new Opponent.
106
107 if (gameTemplate.getOpponentName() != null) {
108
109     String opponentName = gameTemplate.getOpponentName();
110     // (for debugging purposes:)
111     System.out.println(opponentName);
112
113     Optional<Opponent> opponentOptional = opponentRepository.findByOpponentName
114         (opponentName);
115     if (opponentOptional.isPresent()) {
116
117         game.setOpponent(opponentOptional.get());
118     } else {
119         Opponent opponent = new Opponent(opponentName);
120         game.setOpponent(opponentRepository.save(opponent));
121     }
122 }
123 // handle Game Date
124 if (!gameTemplate.getGameDate().isEmpty()) {
125     game.setGameDate(gameTemplate.getGameDate());
126 }
127 // handle Number of Goals Opponent
128 if (gameTemplate.getNbGoalsOpponent() != null) {
129     Integer nbGoalsOpponent = new Integer(gameTemplate.getNbGoalsOpponent());
130     System.out.println(nbGoalsOpponent);
131     game.setNbGoalsOpponent(nbGoalsOpponent);
132 }
133 // handle Number of Goals Keile
134 if (gameTemplate.getNbGoalsKeile() != null) {
135     Integer nbGoalsKeile = new Integer(gameTemplate.getNbGoalsKeile());
136     System.out.println(nbGoalsKeile);
137     game.setNbGoalsKeile(nbGoalsKeile);
138 }
139
140 // handle list of playerId's representing players that participated in the game
141
142 Long[] idsOfPlayersAtGame = gameTemplate.getPlayerIdList();
143 List<Player> playersAtGame = new ArrayList<>();
144
145 if (idsOfPlayersAtGame != null) {
146
147     for (int i = 0; i < idsOfPlayersAtGame.length; i++) {
148
149         // test whether player exists in Database, if yes, load and save it to
150         // List of
151         // players of the game
152         Optional<Player> player = playerRepository.findById(idsOfPlayersAtGame[i
153             ]);
154
155         if (player.isPresent()) {
156             playersAtGame.add(player.get());
157         } else {
158             return ResponseEntity.badRequest().body("player with id " +
159                 idsOfPlayersAtGame[i] + " not found");
160         }
161     }
162 }

```


[illegible]

```
207         if (secondAssistantOptional.isPresent()) {
208
209             Player secondAssistant = secondAssistantOptional.
210                 get();
211             Player firstAssistant = firstAssistantOptional.get
212                 ();
213             Player goalScorer = goalScorerOptional.get();
214
215             Goal goal = new Goal(goalScorer, firstAssistant,
216                 secondAssistant, game);
217             goalRepository.save(goal);
218             goalsAtGame.add(goal);
219
220             if (!playersAtGame.contains(goalScorer)) {
221                 playersAtGame.add(goalScorer);
222             }
223             if (!playersAtGame.contains(firstAssistant)) {
224                 playersAtGame.add(firstAssistant);
225             }
226             if (!playersAtGame.contains(secondAssistant)) {
227                 playersAtGame.add(secondAssistant);
228             }
229             game.setPlayers(playersAtGame);
230         }
231
232         else {
233             return ResponseEntity.status(HttpStatus.
234                 BAD_REQUEST)
235                 .body("Player with id " + secondAssistantId
236                     + " not found");
237         }
238     }
239
240     else {
241         Player firstAssistant = firstAssistantOptional.get();
242         Player goalScorer = goalScorerOptional.get();
243
244         Goal goal = new Goal(goalScorer, firstAssistant, game)
245             ;
246         goalRepository.save(goal);
247         goalsAtGame.add(goal);
248
249         if (!playersAtGame.contains(goalScorer)) {
250             playersAtGame.add(goalScorer);
251         }
252         if (!playersAtGame.contains(firstAssistant)) {
253             playersAtGame.add(firstAssistant);
254         }
255         game.setPlayers(playersAtGame);
256     }
257 } else {
258     return ResponseEntity.status(HttpStatus.BAD_REQUEST)
259         .body("Game contains Goal with invalid first
260             Assistant, Player with id "
261                 + firstAssistantId + " not found");
262 }
```

```

258         else {
259             if (currentGoalTemplate.getSecondAssistantId() != null) {
260
261                 Long secondAssistantId = currentGoalTemplate.
262                     getSecondAssistantId();
263                 Optional<Player> secondAssistantOptional =
264                     playerRepository.findById(secondAssistantId);
265
266                 if (secondAssistantOptional.isPresent()) {
267                     Player secondAssistant = secondAssistantOptional.get()
268                         ;
269                     Player goalScorer = goalScorerOptional.get();
270
271                     Goal goal = new Goal(goalScorer, secondAssistant, game
272                         );
273                     goalRepository.save(goal);
274                     goalsAtGame.add(goal);
275
276                     if (!playersAtGame.contains(goalScorer)) {
277                         playersAtGame.add(goalScorer);
278                     }
279                     if (!playersAtGame.contains(secondAssistant)) {
280                         playersAtGame.add(secondAssistant);
281                     }
282                     game.setPlayers(playersAtGame);
283                 }
284             }
285             else {
286                 return ResponseEntity.status(HttpStatus.BAD_REQUEST)
287                     .body("Game contains Goal with invalid second
288                         Assistant, Player with id "
289                         + secondAssistantId + " not found");
290             }
291         }
292     }
293     else {
294         Player goalScorer = goalScorerOptional.get();
295         Goal goal = new Goal(goalScorer, game);
296         goalRepository.save(goal);
297         goalsAtGame.add(goal);
298
299         if (!playersAtGame.contains(goalScorer)) {
300             playersAtGame.add(goalScorer);
301         }
302         game.setPlayers(playersAtGame);
303     }
304 }
305 }
306 } else {
307     return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
308         "Game contains goal with invalid scorer. Player with id "
309         + scorerId + " not found");
310 }
311 }
312 } else {
313     return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Game
314         contains goal with missing scorer");
315 }
316 }

```

```

309     }
310     game.setGoalsKeile(goalsAtGame);
311 }
312
313 // to return a ResponseEntity with a header containing the URI of the created
314 // resource
315 Game savedGame = gameRepository.save(game);
316 URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{gameId}"
317     ")
318     .buildAndExpand(savedGame.getId()).toUri();
319 return ResponseEntity.created(location).body(game);
320 }
321
322 @PutMapping("/games/{gameId}")
323 public ResponseEntity<Object> updateGame(@RequestBody GameTemplate gameTemplate,
324     @PathVariable("gameId") Long gameId) {
325
326     Optional<Game> gameOptional = gameRepository.findById(gameId);
327
328     if (!gameOptional.isPresent())
329         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("invalid game id"
330             );
331
332     Game game = gameOptional.get();
333     //in order to update List of goals and to cascade changes
334     for(int i = 0; i < game.getGoalsKeile().size(); i++) {
335         System.out.println(game.getGoalsKeile().get(i).getGoalId());
336         goalRepository.delete(game.getGoalsKeile().get(i));
337     }
338
339     // Handle opponent. update opponent if new value provided
340     if (gameTemplate.getOpponentName() != null) {
341
342         String opponentName = gameTemplate.getOpponentName();
343         Optional<Opponent> opponentOptional = opponentRepository.findByOpponentName
344             (opponentName);
345         if (opponentOptional.isPresent()) {
346
347             game.setOpponent(opponentOptional.get());
348         } else {
349             Opponent opponent = new Opponent(opponentName);
350             game.setOpponent(opponentRepository.save(opponent));
351         }
352     }
353     // handle game date
354     if (!gameTemplate.getGameDate().isEmpty()) {
355         game.setGameDate(gameTemplate.getGameDate());
356     }
357     // handle number of goals of the opponent
358     if (gameTemplate.getNbGoalsOpponent() != null) {
359         Integer nbGoalsOpponent = gameTemplate.getNbGoalsOpponent();
360         game.setNbGoalsOpponent(nbGoalsOpponent);
361     }
362     // handle Number of goals of HC Keile
363     if (gameTemplate.getNbGoalsKeile() != null) {
364         Integer nbGoalsKeile = gameTemplate.getNbGoalsKeile();

```

```

364     game.setNbGoalsKeile(nbGoalsKeile);
365 }
366
367 // handle List of playerId's representing players that participated in the game
368 Long[] idsOfPlayersAtGame = gameTemplate.getPlayerIdList();
369 List<Player> playersAtGame = new ArrayList<>();
370
371 if (idsOfPlayersAtGame != null) {
372     for (int i = 0; i < idsOfPlayersAtGame.length; i++) {
373         // test whether player exists in Database, if yes, load and save it to
374         // List of
375         // players of the game
376         Optional<Player> player = playerRepository.findById(idsOfPlayersAtGame[i
377             ]);
378
379         if (player.isPresent()) {
380             playersAtGame.add(player.get());
381         } else {
382             return ResponseEntity.badRequest().body("player with id " +
383                 idsOfPlayersAtGame[i] + " not found");
384         }
385     }
386     game.setPlayers(playersAtGame);
387 }
388
389 // handle list of Goals: Check, whether it has a scorer and zero or one or two
390 // assistants. Create Goal, save it to game.
391
392 List<GoalTemplate> goalTemplateList = gameTemplate.getGoalsList();
393 List<Goal> goalsAtGame = new ArrayList<>();
394
395 if (goalTemplateList != null) {
396     for (int i = 0; i < goalTemplateList.size(); i++) {
397         GoalTemplate currentGoalTemplate = goalTemplateList.get(i);
398         Long scorerId = currentGoalTemplate.getGoalScorerId();
399
400         if (scorerId != null) {
401             Optional<Player> goalScorerOptional = playerRepository.findById(
402                 scorerId);
403
404             if (goalScorerOptional.isPresent()) {
405                 Long firstAssistantId = currentGoalTemplate.getFirstAssistantId()
406                     ;
407
408                 if (firstAssistantId != null) {
409                     if (firstAssistantId.equals(scorerId)) {
410                         return ResponseEntity.status(HttpStatus.BAD_REQUEST)
411                             .body("goalscorer and first assistant cannot be
412                                 the same player");
413                     }
414

```

```

415         Optional<Player> firstAssistantOptional = playerRepository.
            findById(firstAssistantId);
416
417         if (firstAssistantOptional.isPresent()) {
418
419             Long secondAssistantId = currentGoalTemplate.
                getSecondAssistantId();
420
421             if (secondAssistantId != null) {
422                 if (secondAssistantId.equals(firstAssistantId)
423                     || secondAssistantId.equals(scorerId)) {
424                     return ResponseEntity.status(HttpStatus.
                        BAD_REQUEST).body(
425                         "assistant cannot be the same player as
                            other assistant or goalscorer");
426                 }
427
428                 Optional<Player> secondAssistantOptional =
                    playerRepository
                        .findById(secondAssistantId);
429
430                 if (secondAssistantOptional.isPresent()) {
431
432                     Player secondAssistant = secondAssistantOptional.
                        get();
433                     Player firstAssistant = firstAssistantOptional.get
                        ();
434                     Player goalScorer = goalScorerOptional.get();
435
436                     Goal goal = new Goal(goalScorer, firstAssistant,
                        secondAssistant, game);
437                     goalRepository.save(goal);
438                     goalsAtGame.add(goal);
439
440
441                     if (!playersAtGame.contains(goalScorer)) {
442                         playersAtGame.add(goalScorer);
443                     }
444                     if (!playersAtGame.contains(firstAssistant)) {
445                         playersAtGame.add(firstAssistant);
446                     }
447                     if (!playersAtGame.contains(secondAssistant)) {
448                         playersAtGame.add(secondAssistant);
449                     }
450                     game.setPlayers(playersAtGame);
451                 }
452             }
453
454             else {
455                 return ResponseEntity.status(HttpStatus.
                    BAD_REQUEST)
                    .body("Player with id " + secondAssistantId
                        + " not found");
456             }
457         }
458     }
459
460     else {
461         Player firstAssistant = firstAssistantOptional.get();
462         Player goalScorer = goalScorerOptional.get();

```

```

463         Goal goal = new Goal(goalScorer, firstAssistant, game)
464         ;
465         goalsAtGame.add(goal);
466         goalRepository.save(goal);
467
468         if (!playersAtGame.contains(goalScorer)) {
469             playersAtGame.add(goalScorer);
470         }
471         if (!playersAtGame.contains(firstAssistant)) {
472             playersAtGame.add(firstAssistant);
473         }
474         game.setPlayers(playersAtGame);
475     }
476 } else {
477     return ResponseEntity.status(HttpStatus.BAD_REQUEST)
478         .body("Game contains Goal with invalid first
479             Assistant, Player with id "
480                 + firstAssistantId + " not found");
481 }
482
483 else {
484     if (currentGoalTemplate.getSecondAssistantId() != null) {
485
486         Long secondAssistantId = currentGoalTemplate.
487             getSecondAssistantId();
488         Optional<Player> secondAssistantOptional =
489             playerRepository.findById(secondAssistantId);
490
491         if (secondAssistantOptional.isPresent()) {
492
493             Player secondAssistant = secondAssistantOptional.get()
494             ;
495             Player goalScorer = goalScorerOptional.get();
496
497             Goal goal = new Goal(goalScorer, secondAssistant, game
498                 );
499             goalsAtGame.add(goal);
500             goalRepository.save(goal);
501
502             if (!playersAtGame.contains(goalScorer)) {
503                 playersAtGame.add(goalScorer);
504             }
505             if (!playersAtGame.contains(secondAssistant)) {
506                 playersAtGame.add(secondAssistant);
507             }
508             game.setPlayers(playersAtGame);
509         }
510     }
511     else {
512         return ResponseEntity.status(HttpStatus.BAD_REQUEST)
513             .body("Game contains Goal with invalid second
514                 Assistant, Player with id "
515                     + secondAssistantId + " not found");
516     }
517 }

```

```

514         else {
515             Player goalScorer = goalScorerOptional.get();
516             Goal goal = new Goal(goalScorer, game);
517             goalRepository.save(goal);
518             goalsAtGame.add(goal);
519
520             if (!playersAtGame.contains(goalScorer)) {
521                 playersAtGame.add(goalScorer);
522             }
523             game.setPlayers(playersAtGame);
524
525         }
526     }
527     } else {
528         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
529             "Game contains goal with invalid scorer. Player with id "
530             + scorerId + " not found");
531     }
532     } else {
533         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Game
534             contains goal with missing scorer");
535     }
536 }
537
538 // to return a ResponseEntity with a header containing the URI of the created
539 // resource
540 Game savedGame = gameRepository.save(game);
541 URI location = ServletUriComponentsBuilder.fromCurrentRequest().buildAndExpand(
542     savedGame.getId()).toUri();
543 return ResponseEntity.created(location).body(game);
544 }
545
546 // Return list of players that participated in a certain game
547 @GetMapping("/games/{gameId}/players")
548 public ResponseEntity<Object> getPlayersOfGame(@PathVariable("gameId") Long gameId)
549 {
550     if (gameRepository.findById(gameId).isPresent()) {
551
552         List<Player> players = gameRepository.findById(gameId).get().getPlayers();
553         if (!players.isEmpty()) {
554             return new ResponseEntity<>(players, HttpStatus.OK);
555         } else {
556             return ResponseEntity.status(HttpStatus.OK).body("No players saved for
557                 game with id " + gameId);
558         }
559     }
560
561     return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Game with id " +
562         gameId + " not found");
563 }

```


563 }

Listing B.16: Klasse GameController vollständig

```

1 package ch.keilestats.app.controller;
2
3
4 import java.net.URI;
5 import java.util.Optional;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.dao.EmptyResultDataAccessException;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.http.ResponseEntity;
11 import org.springframework.web.bind.annotation.DeleteMapping;
12 import org.springframework.web.bind.annotation.GetMapping;
13 import org.springframework.web.bind.annotation.PathVariable;
14 import org.springframework.web.bind.annotation.PostMapping;
15 import org.springframework.web.bind.annotation.PutMapping;
16 import org.springframework.web.bind.annotation.RequestBody;
17 import org.springframework.web.bind.annotation.RequestMapping;
18 import org.springframework.web.bind.annotation.RestController;
19 import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
20
21 import ch.keilestats.app.datatemplates.GoalTemplate;
22 import ch.keilestats.app.entities.Goal;
23 import ch.keilestats.app.entities.Player;
24 import ch.keilestats.app.repositories.GoalRepository;
25 import ch.keilestats.app.repositories.PlayerRepository;
26
27 @RestController
28 /*
29  * A convenience annotation that is itself annotated with @Controller
30  * and @ResponseBody. @Controller is a specification of @Component to indicate
31  * to the framework that the container should instantiate the class as a bean
32  */
33 @RequestMapping("/app")
34 /*
35  * Annotation for mapping web requests onto methods in request-handling classes
36  * with flexible method signatures
37  */
38 public class GoalController { // handles calls on goal resources
39
40     @Autowired /*
41         * Marks a constructor, field, setter method, or config method as to be
42         * autowired by Spring's dependency injection facilities.
43         */
44     private GoalRepository goalRepository;
45     @Autowired
46     private PlayerRepository playerRepository;
47
48     // handles GET-Requests on the "/goals" resource
49     @GetMapping("/goals")
50     public ResponseEntity<Object> getAllGoals() {
51
52         return ResponseEntity.status(HttpStatus.OK).body(goalRepository.findAll());

```

```

53     }
54
55     @GetMapping("/goals/{goalId}")
56     public ResponseEntity<Object> getGoalById(@PathVariable("goalId") Long goalId) {
57
58         Optional<Goal> optionalGoal = goalRepository.findById(goalId);
59         if (optionalGoal.isPresent()) {
60             return new ResponseEntity<>(optionalGoal.get(), HttpStatus.OK);
61         } else {
62             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("goal not found")
63                 ;
64         }
65     }
66
67     /*
68     * Returning a ResponseEntity with a header containing the URL of the created
69     * resource. @RequestBody indicates that the Body of the Request should be bound
70     * to the Goal object, not some Header Parameters
71     */
72     @PostMapping(path = "/goals")
73     public ResponseEntity<Object> addGoal(@RequestBody GoalTemplate goalTemplate) {
74
75         Goal goal = new Goal();
76
77         /*
78         * Check whether id's of goal scorers and assistants are valid. Check whether
79         * values are not null. Create goal with one, two or no assist accordingly.
80         * Assure that goal scorer and assistant or assistants are not the same player
81         */
82         if (goalTemplate.getGoalScorerId() != null) {
83
84             if (playerRepository.findById(goalTemplate.getGoalScorerId()).isPresent())
85             {
86                 Optional<Player> goalScorerOptional = playerRepository.findById(
87                     goalTemplate.getGoalScorerId());
88                 Player goalScorer = goalScorerOptional.get();
89                 goal.setGoalScorer(goalScorer);
90             } else {
91                 return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("invalid data:
92                     unknown player");
93             }
94
95             if (goalTemplate.getFirstAssistantId() != null) {
96
97                 if (playerRepository.findById(goalTemplate.getFirstAssistantId()).
98                     isPresent()
99                     && !goalTemplate.getFirstAssistantId().equals(goalTemplate.
100                         getGoalScorerId())) {
101                     Optional<Player> firstAssistantOptional = playerRepository
102                         .findById(goalTemplate.getFirstAssistantId());
103                     Player firstAssistant = firstAssistantOptional.get();
104                     goal.setFirstAssistant(firstAssistant);
105                 }
106
107                 else {
108                     return ResponseEntity.status(HttpStatus.BAD_REQUEST)
109                         .body("invalid data: unknown or identical players provided");
110                 }
111             }
112         }

```

```

105         if (goalTemplate.getSecondAssistantId() != null) {
106
107             if (playerRepository.findById(goalTemplate.getSecondAssistantId()).
108                 isPresent()
109                 && !(goalTemplate.getSecondAssistantId().equals(goalTemplate.
110                     getFirstAssistantId())
111                     || goalTemplate.getSecondAssistantId().equals(
112                         goalTemplate.getGoalScorerId())) {
113                 Optional<Player> secondAssistantOptional = playerRepository
114                     .findById(goalTemplate.getSecondAssistantId());
115                 Player secondAssistant = secondAssistantOptional.get();
116                 goal.setSecondAssistant(secondAssistant);
117             } else {
118                 return ResponseEntity.status(HttpStatus.BAD_REQUEST)
119                     .body("invalid data: identical or unknown player id's");
120             }
121         } else {
122             if (goalTemplate.getSecondAssistantId() != null) {
123
124                 if (playerRepository.findById(goalTemplate.getSecondAssistantId()).
125                     isPresent()
126                     && !goalTemplate.getSecondAssistantId().equals(goalTemplate.
127                         getGoalScorerId())) {
128
129                     Optional<Player> secondAssistantOptional = playerRepository
130                         .findById(goalTemplate.getSecondAssistantId());
131                     Player secondAssistant = secondAssistantOptional.get();
132                     goal.setSecondAssistant(secondAssistant);
133                 } else {
134                     return ResponseEntity.status(HttpStatus.BAD_REQUEST)
135                         .body("invalid data: identical or unknown player id's");
136                 }
137             }
138         }
139         Goal savedGoal = goalRepository.save(goal);
140         URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{
141             goalId}")
142             .buildAndExpand(savedGoal.getGoalId()).toUri();
143
144         return ResponseEntity.created(location).body(savedGoal);
145     } else {
146         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("goal scorer
147             missing");
148     }
149 }
150
151 @PutMapping("/goals/{goalId}")
152 public ResponseEntity<Object> updateGoal(@RequestBody GoalTemplate goalTemplate,
153     @PathVariable("goalId") Long goalId) {
154
155     /*
156      * In addition to the checks in the post method, assure that updated goal
157      * scorer
158      * or assistant is not identical with value in the existing resource
159      */
160     Goal goal;

```

```

155     if (goalRepository.findById(goalId).isPresent())
156         goal = goalRepository.findById(goalId).get();
157     else
158         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("goal to update
            not found");
159
160     Long scorerId = goalTemplate.getGoalScorerId();
161     Long firstAssistantId = goalTemplate.getFirstAssistantId();
162     Long secondAssistantId = goalTemplate.getSecondAssistantId();
163
164     // handle different cases (none, one, two or three null values), validation of
165     // values
166     if (scorerId == null && firstAssistantId == null && secondAssistantId == null)
167         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("no values to
            update provided");
168
169     else if (scorerId != null && firstAssistantId != null && secondAssistantId !=
        null) {
170         if (scorerId.equals(firstAssistantId) || scorerId.equals(secondAssistantId)
            || firstAssistantId.equals(secondAssistantId)) {
171             return ResponseEntity.status(HttpStatus.BAD_REQUEST)
172                 .body("assistant and goal scorer or first and second assistant
                    cannot be the same player");
173         }
174     }
175
176     if (!playerRepository.findById(scorerId).isPresent()
        || !playerRepository.findById(firstAssistantId).isPresent()
        || !playerRepository.findById(secondAssistantId).isPresent()) {
177         return ResponseEntity.status(HttpStatus.BAD_REQUEST)
178             .body("invalid playerId provided: player does not exist");
179     } else {
180         goal.setGoalScorer(playerRepository.findById(scorerId).get());
181         goal.setFirstAssistant(playerRepository.findById(firstAssistantId).get()
            );
182         goal.setSecondAssistant(playerRepository.findById(secondAssistantId).get()
            );
183         goalRepository.save(goal);
184         return ResponseEntity.status(HttpStatus.OK).body(goal);
185     }
186 } else if (scorerId == null) {
187     if (firstAssistantId == null) {
188         if (playerRepository.findById(secondAssistantId).isPresent()) {
189             goal.setSecondAssistant(playerRepository.findById(secondAssistantId).
190                 get());
191             goalRepository.save(goal);
192             return ResponseEntity.status(HttpStatus.OK).body(goal);
193         } else
194             return ResponseEntity.status(HttpStatus.BAD_REQUEST)
195                 .body("invalid playerId provided: player with id " +
                    secondAssistantId + " does not exist");
196     } else {
197         if (secondAssistantId == null) {
198             if (playerRepository.findById(firstAssistantId).isPresent()) {
199                 if (!goal.getSecondAssistant().getPlayerId().equals(
200                     firstAssistantId)
201                     && !goal.getGoalScorer().getPlayerId().equals(
                    firstAssistantId)) {

```

```

202         goal.setFirstAssistant(playerRepository.findById(
203             firstAssistantId).get());
204         goalRepository.save(goal);
205         return ResponseEntity.status(HttpStatus.OK).body(goal);
206     } else
207         return ResponseEntity.status(HttpStatus.BAD_REQUEST)
208             .body("two assistants or assistant and goal scorer
209                 cannot be the same player");
210     } else
211         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
212             "invalid playerId provided: player with id " +
213             firstAssistantId + " does not exist");
214     } else {
215         if (!playerRepository.findById(firstAssistantId).isPresent()
216             || !playerRepository.findById(secondAssistantId).isPresent())
217             {
218                 return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("
219                     invalid playerId provided");
220             }
221         if (!firstAssistantId.equals(secondAssistantId)
222             && !(goal.getGoalScorer().getPlayerId().equals(
223                 firstAssistantId)
224                 || goal.getGoalScorer().getPlayerId().equals(
225                     secondAssistantId))) {
226             goal.setFirstAssistant(playerRepository.findById(
227                 firstAssistantId).get());
228             goal.setSecondAssistant(playerRepository.findById(
229                 secondAssistantId).get());
230             goalRepository.save(goal);
231             return ResponseEntity.status(HttpStatus.OK).body(goal);
232         } else {
233             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
234                 "first and second assistant or assistant and goal scorer
235                 cannot be the same player");
236         }
237     }
238 }
239 } else { // meaning if scorerId != null
240     if (firstAssistantId == null && secondAssistantId == null) {
241         if (!playerRepository.findById(scorerId).isPresent())
242             return ResponseEntity.status(HttpStatus.BAD_REQUEST)
243                 .body("invalid playerId provided player with id " + scorerId
244                     + " does not exist");
245         else {
246             if (!(goal.getFirstAssistant().getPlayerId().equals(scorerId)
247                 || goal.getSecondAssistant().getPlayerId().equals(scorerId)))
248                 {
249                     goal.setGoalScorer(playerRepository.findById(scorerId).get());
250                     goalRepository.save(goal);
251                     return ResponseEntity.status(HttpStatus.OK).body(goal);
252                 } else
253                     return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
254                         "first and second assistant or assitant and goal scorer
255                         cannot be the same player");
256             }
257         } else {
258             if (firstAssistantId == null) {
259                 if (!playerRepository.findById(scorerId).isPresent())

```

```

247         || !playerRepository.findById(secondAssistantId).isPresent())
248         {
249             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("
250                 invalid playerId provided");
251         } else {
252             if (!(secondAssistantId.equals(scorerId)
253                 || goal.getFirstAssistant().getPlayerId().equals(
254                     secondAssistantId))) {
255                 goal.setSecondAssistant(playerRepository.findById(
256                     secondAssistantId).get());
257                 goal.setGoalScorer(playerRepository.findById(scorerId).get())
258                 ;
259                 goalRepository.save(goal);
260                 return ResponseEntity.status(HttpStatus.OK).body(goal);
261             } else {
262                 return ResponseEntity.status(HttpStatus.BAD_REQUEST)
263                     .body("goal scorer and assistant or two assistants
264                         cannot be the same player");
265             }
266         }
267     } else { // second assistant == null
268         if (!playerRepository.findById(scorerId).isPresent()
269             || !playerRepository.findById(firstAssistantId).isPresent())
270         {
271             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("
272                 invalid playerId provided");
273         } else {
274             if (!(firstAssistantId.equals(scorerId)
275                 || scorerId.equals(goal.getSecondAssistant().getPlayerId
276                     ()))
277                 || firstAssistantId.equals(goal.getSecondAssistant().
278                     getPlayerId())) {
279                 goal.setFirstAssistant(playerRepository.findById(
280                     firstAssistantId).get());
281                 goal.setGoalScorer(playerRepository.findById(scorerId).get())
282                 ;
283                 goalRepository.save(goal);
284                 return ResponseEntity.status(HttpStatus.OK).body(goal);
285             } else {
286                 return ResponseEntity.status(HttpStatus.BAD_REQUEST)
287                     .body("goal scorer and assistant cannot be the same
288                         player");
289             }
290         }
291     }
292 }
293
294 @DeleteMapping("/goals/{goalId}")
295 public ResponseEntity<Object> deleteGoal(@PathVariable("goalId") Long goalId) {
296     try {
297         goalRepository.deleteById(goalId);
298         return ResponseEntity.status(HttpStatus.OK).body("goal with id " + goalId +
299             " deleted");
300     } catch (EmptyResultDataAccessException e) {
301         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("goal not found")
302         ;
303     }
304 }

```

```

290     }
291 }
292 }

```

Listing B.17: Klasse GoalController vollständig

```

1 package ch.keilestats.app.controller;
2
3 import java.net.URI;
4 import java.util.Optional;
5
6 import javax.validation.Valid;
7
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.http.MediaType;
11 import org.springframework.http.ResponseEntity;
12 import org.springframework.web.bind.annotation.DeleteMapping;
13 import org.springframework.web.bind.annotation.GetMapping;
14 import org.springframework.web.bind.annotation.PathVariable;
15 import org.springframework.web.bind.annotation.PostMapping;
16 import org.springframework.web.bind.annotation.PutMapping;
17 import org.springframework.web.bind.annotation.RequestBody;
18 import org.springframework.web.bind.annotation.RequestMapping;
19 import org.springframework.web.bind.annotation.RestController;
20 import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
21
22 import ch.keilestats.app.datatemplates.OpponentTemplate;
23 import ch.keilestats.app.entities.Opponent;
24 import ch.keilestats.app.repositories.OpponentRepository;
25
26 @RestController
27 /*
28  * A convenience annotation that is itself annotated with @Controller
29  * and @ResponseBody. @Controller is a specification of @Component to indicate
30  * to the framework that the container should instantiate the class as a bean
31  */
32 @RequestMapping("/app")
33 /*
34  * Annotation for mapping web requests onto methods in request-handling classes
35  * with flexible method signatures
36  */
37 public class OpponentController {
38
39     @Autowired
40     /*
41      * Marks a constructor, field, setter method, or config method as to be
42      * autowired by Spring's dependency injection facilities.
43      */
44     private OpponentRepository opponentRepository;
45
46     @GetMapping("/opponents")
47     public ResponseEntity<Object> getAllOpponents() {
48
49         return ResponseEntity.status(HttpStatus.OK).body(opponentRepository.findAll());
50     }

```

```

51
52 @GetMapping("/opponents/{opponentId}")
53 public ResponseEntity<Object> getOpponentById(@PathVariable("opponentId") Long
    opponentId) {
54
55     if (!opponentRepository.findById(opponentId).isPresent()) {
56         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("No Opponent with
            id " + opponentId + " found");
57     }
58
59     else {
60         return new ResponseEntity<>(opponentRepository.findById(opponentId).get(),
            HttpStatus.OK);
61     }
62 }
63
64 @DeleteMapping("/opponents/{opponentId}")
65 public ResponseEntity<Object> deleteOpponent(@PathVariable("opponentId") Long
    opponentId) {
66
67     Optional<Opponent> opponentOptional = opponentRepository.findById(opponentId);
68     if (opponentOptional.isPresent()) {
69         // checks, if opponent is already saved in a game. if yes. opponent shall
            not be
70         // deleted.
71         if (opponentOptional.get().getGames().isEmpty()) {
72             opponentRepository.deleteById(opponentId);
73             return ResponseEntity.status(HttpStatus.OK).body("Opponent deleted");
74         }
75
76         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(
77             "Opponent with id " + opponentId + " has a non-empty list of games
                and therefore cannot be deleted");
78     }
79
80     return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Opponent with id " +
        opponentId + " not found");
81 }
82
83 @PostMapping(path = "/opponents", consumes = { MediaType.APPLICATION_JSON_VALUE, "
    application/json" })
84 public ResponseEntity<Object> addOpponent(@Valid @RequestBody OpponentTemplate
    opponentTemplate) {
85
86     if (opponentTemplate.getOpponentName() != null && !opponentTemplate.
        getOpponentName().isEmpty()) {
87
88         String opponentName = opponentTemplate.getOpponentName();
89
90         Optional<Opponent> opponentOptional = opponentRepository.findByOpponentName
            (opponentName);
91
92         if (!opponentOptional.isPresent()) {
93             Opponent opponent = new Opponent(opponentName);
94             Opponent savedOpponent = opponentRepository.save(opponent);
95             URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{
                opponentId}")
96                 .buildAndExpand(savedOpponent.getOpponentId()).toUri();

```



```

97         return ResponseEntity.status(HttpStatus.CREATED).location(location).body
98             (savedOpponent);
99     }
100
101     else {
102         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Opponent
103             already exists");
104     }
105 }
106 return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("No opponent name
107     provided");
108 }
109
110 @PutMapping(path = "/opponents/{opponentId}")
111 public ResponseEntity<Object> updateOpponent(@RequestBody OpponentTemplate
112     opponentTemplate,
113     @PathVariable("opponentId") Long opponentId) {
114
115     Optional<Opponent> opponentOptional = opponentRepository.findById(opponentId);
116     if (!opponentOptional.isPresent())
117         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Opponent with id
118             " + opponentId + " not found");
119
120     if (opponentTemplate.getOpponentName() != null)
121         opponentOptional.get().setOpponentName(opponentTemplate.getOpponentName());
122
123     URI location = ServletUriComponentsBuilder.fromCurrentRequest()
124         .buildAndExpand(opponentOptional.get().getOpponentId()).toUri();
125
126     opponentRepository.save(opponentOptional.get());
127     return ResponseEntity.status(HttpStatus.OK).location(location).body(
128         opponentOptional.get());
129 }

```

Listing B.18: Klasse OpponentController vollständig

```

1 package ch.keilestats.app.controller;
2
3 import ch.keilestats.app.datatemplates.PlayerTemplate;
4 import ch.keilestats.app.datatemplates.ScoringDataTemplate;
5 import ch.keilestats.app.entities.*;
6 import ch.keilestats.app.repositories.*;
7
8 import java.net.URI;
9 import java.util.ArrayList;
10 import java.util.List;
11 import java.util.Optional;
12
13 import org.springframework.beans.factory.annotation.Autowired;
14 import org.springframework.http.HttpHeaders;
15 import org.springframework.http.HttpStatus;
16 import org.springframework.http.ResponseEntity;

```

```

17 import org.springframework.web.bind.annotation.DeleteMapping;
18 import org.springframework.web.bind.annotation.GetMapping;
19 import org.springframework.web.bind.annotation.PathVariable;
20 import org.springframework.web.bind.annotation.PostMapping;
21 import org.springframework.web.bind.annotation.PutMapping;
22 import org.springframework.web.bind.annotation.RequestBody;
23 import org.springframework.web.bind.annotation.RequestMapping;
24 import org.springframework.web.bind.annotation.RestController;
25 import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
26
27 @RestController
28 /*
29  * A convenience annotation that is itself annotated with @Controller
30  * and @ResponseBody. @Controller is a specification of @Component to indicate
31  * to the framework that the container should instantiate the class as a bean
32  */
33 @RequestMapping("/app")
34 /*
35  * Annotation for mapping web requests onto methods in request-handling classes
36  * with flexible method signatures
37  */
38 public class PlayerController {
39
40     @Autowired /* Annotation to tell the framework to inject this dependency */
41     private PlayerRepository playerRepository;
42
43     // Return list of all players
44     @GetMapping("/players")
45     public ResponseEntity<Object> getAllPlayers() {
46
47         return ResponseEntity.status(HttpStatus.OK).body(playerRepository.findAll());
48     }
49
50     @DeleteMapping("/players/{playerId}")
51     public ResponseEntity<Object> deletePlayer(@PathVariable("playerId") Long playerId)
52     {
53
54         if (playerRepository.findById(playerId).isPresent()) {
55
56             Player playerToBeDeleted = playerRepository.findById(playerId).get();
57
58             if (playerToBeDeleted.getGames().isEmpty()) {
59                 playerRepository.deleteById(playerId);
60                 return new ResponseEntity<>("Player deleted..", HttpStatus.OK);
61             } else
62                 return new ResponseEntity<>("Player with id: " + playerId + " has a non-
63                     empty List of games "
64                     + "and therefore cannot be deleted", HttpStatus.BAD_REQUEST);
65         }
66
67         return new ResponseEntity<>("Player with id " + playerId + " not found",
68             HttpStatus.BAD_REQUEST);
69     }
70
71     // Return values of one Player
72     @GetMapping("/players/{playerId}")
73     public ResponseEntity<Object> getPlayerById(@PathVariable("playerId") Long
74         playerId) {

```

```

71     Optional<Player> playerOptional = playerRepository.findById(playerId);
72
73     if (playerOptional.isPresent()) {
74         return ResponseEntity.status(HttpStatus.OK).body(playerOptional.get());
75     } else {
76         return ResponseEntity.status(HttpStatus.NOT_FOUND).body("player with id " +
77             playerId + " not found");
78     }
79 }
80
81 // Returning a ResponseEntity with a header containing the URL of the created
82 // resource
83 @PostMapping(path = "/players")
84 public ResponseEntity<Object> addPlayer(@RequestBody PlayerTemplate playerTemplate)
85 {
86     Player player = new Player();
87
88     player.setFirstname(playerTemplate.getFirstname());
89     player.setLastname(playerTemplate.getLastname());
90     player.setAddress(playerTemplate.getAddress());
91     player.setPosition(playerTemplate.getPosition());
92     player.setEmail(playerTemplate.getEmail());
93     player.setPhone(playerTemplate.getPhone());
94
95     if (playerRepository.findAll().contains(player)) {
96         return new ResponseEntity<>(
97             "Player with name \"\" + player.getFirstname() + \" \" + player.
98             getLastName() + \"\" already exists",
99             HttpStatus.BAD_REQUEST);
100     }
101
102     Player savedPlayer = playerRepository.save(player);
103
104     URI location = ServletUriComponentsBuilder.fromCurrentRequest().path("/{
105         playerId}")
106         .buildAndExpand(savedPlayer.getPlayerId()).toUri();
107
108     return ResponseEntity.created(location).body(savedPlayer);
109 }
110
111 @PutMapping("/players/{playerId}")
112 public ResponseEntity<Object> updatePlayer(@RequestBody PlayerTemplate
113     playerTemplate,
114     @PathVariable("playerId") Long playerId) {
115
116     HttpHeaders headers = new HttpHeaders();
117     headers.add("status1", "new player created or player " + playerId + " updated")
118     ;
119
120     // Check whether playerId exists, update player, or else create new one
121     return new ResponseEntity<Object>(playerRepository.findById(playerId).map(
122         player -> {
123             if (playerTemplate.getAddress() != null)
124                 player.setAddress(playerTemplate.getAddress());

```

```

122         if (playerTemplate.getEmail() != null) {
123             if (playerTemplate.getEmail() != null)
124                 player.setEmail(playerTemplate.getEmail());
125         }
126         if (playerTemplate.getFirstname() != null)
127             player.setFirstname(playerTemplate.getFirstname());
128         if (playerTemplate.getLastname() != null)
129             player.setLastname(playerTemplate.getLastname());
130         if (playerTemplate.getPhone() != null)
131             player.setPhone(playerTemplate.getPhone());
132         if (playerTemplate.getPosition() != null)
133             player.setEmail(playerTemplate.getPosition());
134         return playerRepository.save(player);
135     }).orElseGet(() -> {
136         Player player = new Player();
137         if (playerTemplate.getAddress() != null)
138             player.setAddress(playerTemplate.getAddress());
139         if (playerTemplate.getEmail() != null)
140             player.setEmail(playerTemplate.getEmail());
141         if (playerTemplate.getFirstname() != null)
142             player.setFirstname(playerTemplate.getFirstname());
143         if (playerTemplate.getLastname() != null)
144             player.setLastname(playerTemplate.getLastname());
145         if (playerTemplate.getPhone() != null)
146             player.setPhone(playerTemplate.getPhone());
147         if (playerTemplate.getPosition() != null)
148             player.setPosition(playerTemplate.getPosition());
149         if (playerId != null)
150             player.setPlayerId(playerId);
151         return playerRepository.save(player);
152     }), headers, HttpStatus.OK);
153 }
154
155 // Return goals where player is goalScorer (or number of Goals)
156 @GetMapping("/palyers/{player_id}/goals")
157 public ResponseEntity<Object> getGoalsByPlayerId(@PathVariable("player_id") Long
158     playerId) {
159
160     int result = 0;
161
162     Optional<Player> optionalPlayer = playerRepository.findById(playerId);
163
164     if (optionalPlayer.isPresent()) {
165         result = optionalPlayer.get().getGoalsScored().size();
166         return ResponseEntity.status(HttpStatus.OK).body(result);
167     } else {
168         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Player with id "
169             + playerId + " not found");
170     }
171 }
172
173 // Return goals, where player did the first assist
174 @GetMapping("/palyers/{player_id}/assists")
175 public ResponseEntity<Object> getPlayerAssistsByPlayerId(@PathVariable("player_id")
176     Long playerId) {

```

```

177     Optional<Player> optionalPlayer = playerRepository.findById(playerId);
178     if (optionalPlayer.isPresent()) {
179         result = optionalPlayer.get().getFirstAssists().size() + optionalPlayer.get
180             ().getSecondAssists().size();
181         return ResponseEntity.status(HttpStatus.OK).body(result);
182     } else {
183         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Player with id "
184             + playerId + " not found");
185     }
186 }
187
188 // Return number of games that a player played
189 @GetMapping("/palyers/{player_id}/games")
190 public ResponseEntity<Object> getGamesByPlayerId(@PathVariable("player_id") Long
191     playerId) {
192
193     int result = 0;
194
195     Optional<Player> optionalPlayer = playerRepository.findById(playerId);
196
197     if (optionalPlayer.isPresent()) {
198         result = optionalPlayer.get().getGames().size();
199         return ResponseEntity.status(HttpStatus.OK).body(result);
200     } else {
201         return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Player with id "
202             + playerId + " not found");
203     }
204 }
205
206 // Return complete scoringtable
207 @GetMapping("/palyers/scoringtable")
208 public ResponseEntity<Object> getScoringdataForEachPlayer() {
209
210     List<ScoringDataTemplate> scoringtable = new ArrayList<>();
211
212     List<Player> allPlayers = playerRepository.findAll();
213
214     for (int i = 0; i < allPlayers.size(); i++) {
215
216         ScoringDataTemplate entry = new ScoringDataTemplate();
217         Player player = allPlayers.get(i);
218
219         entry.setAssistsScored(player.getFirstAssists().size() + player.
220             getSecondAssists().size());
221         entry.setPlayerFirstName(player.getFirstname());
222         entry.setGamesPlayed(player.getGames().size());
223         entry.setGoalsScored(player.getGoalsScored().size());
224         entry.setPlayerLastName(player.getLastname());
225         entry.setPlayerId(player.getPlayerId());
226         entry.setTotalPoints(entry.getAssistsScored() + entry.getGoalsScored());
227
228         scoringtable.add(entry);
229     }
230
231     return ResponseEntity.status(HttpStatus.OK).body(scoringtable);
232 }

```

229 }

Listing B.19: Klasse PlayerController vollständig

B.7. pom.xml-File

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/
      maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6     <parent>
7         <groupId>org.springframework.boot</groupId>
8         <artifactId>spring-boot-starter-parent</artifactId>
9         <version>2.2.0.RELEASE</version>
10        <relativePath /> <!-- lookup parent from repository -->
11    </parent>
12    <groupId>ch.keilestats</groupId>
13    <artifactId>api</artifactId>
14    <version>0.0.1-SNAPSHOT</version>
15    <name>api</name>
16    <description>restful api for database of a icehockeyteam</description>
17
18    <properties>
19        <java.version>1.8</java.version>
20    </properties>
21
22    <dependencies>
23        <dependency>
24            <groupId>org.springframework.boot</groupId>
25            <artifactId>spring-boot-starter-data-jpa</artifactId>
26        </dependency>
27        <dependency>
28            <groupId>org.springframework.boot</groupId>
29            <artifactId>spring-boot-starter-web</artifactId>
30        </dependency>
31
32
33        <dependency>
34            <groupId>com.h2database</groupId>
35            <artifactId>h2</artifactId>
36        </dependency>
37
38        <!-- Spring Boot developer tools. such as automatic restart of server when
39            classes change -->
40        <dependency>
41            <groupId>org.springframework.boot</groupId>
42            <artifactId>spring-boot-devtools</artifactId>
43            <optional>true</optional>
44        </dependency>
45
46        <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
47        <dependency>
48            <groupId>io.springfox</groupId>

```

```

49     <artifactId>springfox-swagger2</artifactId>
50     <version>2.9.2</version>
51 </dependency>
52
53 <!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
54 <dependency>
55     <groupId>io.springfox</groupId>
56     <artifactId>springfox-swagger-ui</artifactId>
57     <version>2.9.2</version>
58 </dependency>
59
60 <dependency>
61     <groupId>org.springframework.boot</groupId>
62     <artifactId>spring-boot-starter-test</artifactId>
63     <scope>test</scope>
64 </dependency>
65
66 <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core
67 -->
68 <dependency>
69     <groupId>com.fasterxml.jackson.core</groupId>
70     <artifactId>jackson-core</artifactId>
71 </dependency>
72 <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
73 databind -->
74 <dependency>
75     <groupId>com.fasterxml.jackson.core</groupId>
76     <artifactId>jackson-databind</artifactId>
77 </dependency>
78
79 </dependencies>
80
81 <build>
82     <plugins>
83         <plugin>
84             <groupId>org.springframework.boot</groupId>
85             <artifactId>spring-boot-maven-plugin</artifactId>
86         </plugin>
87     </plugins>
88 </build>
89
90 </project>

```

Listing B.20: pom.xml-File

B.8. ReadMe-File

```

1 # keillestats-app
2 This project contains a REST API for Statistics of a just-for-fun ice hockey team. It
   is build using Spring Boot -
3 The simple Webservice is intended to be used by Clients to save and read Game-
   Statistics of a just-for-fun Ice-Hockey-Team to and from a database.
4 To run the Webservice:
5
6 1. Import the project into your IDE / on your machine as a Maven project

```

```
7 2. All the dependencies and an embeded Tomcat-Webserver should be imported
   automatically, if not, compile and build the pom.xml-file with Maven again.
8 3. A h2 in-Memory Database is used for the persistence layer. It can be accessed via:
   "http://localhost:8080/h2-console"
9 (It is easy to change the persistence layer. To configure the connection to the
   database, open the "application.properties" file in the source/main/resources
   folder. There, you must change the url, username, password according to your
   database credentials. The dependency to the corresponding database has to be set
   in the pom.xml file)
10 4. Launch the application by running the class KeileStatsApplication in folder
11 src/main/java
12 5. To test the endpoints of the API, go to http://localhost8080/swagger-ui.html. A
   Swagger representation of the project should have been generated and be accessible
   over this address. There, the endpoints are presented and can be tested.
```

Listing B.21: ReadMe-File

C Lizenz

Copyright (c) 2020 Marc Raemy.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

The GNU Free Documentation Licence can be read from [20].

Literaturverzeichnis

- [1] Christian Bauer and Gavin King. *Hibernate in Action*. Manning, 2005. 23, 30
- [2] Sun Microsystems Inc. *JavaBeans*. Oracle, 1997. 26
- [3] Rod Johnson. *Expert One-on-One J2EE Design and Development*. Wrox, 2002. 25
- [4] Rod Johnson, Jürgen Hoeller, et al. Spring 2.0.8, java/j2ee Application Framework - Reference Documentation, 2007. <https://docs.spring.io/spring/docs/2.0.x/spring-reference.pdf>. 25
- [5] Jacques Pasquier, Arnaud Durand, and Gremaud Pascal. Lecture notes advanced software engineering, October 2016. Departement für Informatik, Universität Freiburg (CH). 31
- [6] Michael Simons. *Spring Boot 2 - Moderne Softwareentwicklung mit Spring 5*. dpunkt.verlag, 2018. 40
- [7] Stefan Tilkov, Eigenbrodt Martin, Silvia Schreier, and Wolf Oliver. *REST und HTTP - Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt.verlag, 2015. 24, 31, 32
- [8] Craig Walls. *Spring im Einsatz*. Hansen, 2012. 25, 26, 27, 29, 30
- [9] Craig Walls. *Spring in Action*. Manning, 2019. 27, 29, 30, 44, 45, 47, 48

Webreferenzen

- [10] What is the Spring Framework really all about?, Java Brains. <https://www.youtube.com/watch?v=gq4S-ovWV1M> (Letzter Aufruf Juni 30, 2019).
- [11] Annotations. <https://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html> (Letzter Aufruf August 5, 2019).
- [12] Blog Baeldung, Mapping Entity Class Names to SQL Table Names with JPA. <https://www.baeldung.com/jpa-entity-table-names> (Letzter Aufruf August 1, 2020). 38
- [13] Understanding the Basics of Spring vs. Spring Boot. <https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot> (Letzter Aufruf August 6, 2019).
- [14] Spring Boot Tutorial, How to Do in Java. <https://howtodoinjava.com/spring-boot-tutorials> (Letzter Aufruf August 27, 2019).
- [15] A Comparison Between Spring and Spring Boot. <https://www.baeldung.com/spring-vs-spring-boot> (Letzter Aufruf August 6, 2019).
- [16] Object Messages and Dependencies, YouTube-Kanal Knowledge Dose. <https://www.youtube.com/watch?v=W21CLd9zm9k> (Letzter Aufruf Sept 17, 2019).
- [17] Understanding Dependency Injection, YouTube-Kanal Java Brains. <https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot> (Letzter Aufruf August 6, 2019).
- [18] Difference between Spring and Spring Boot, Dzone. <https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot> (Letzter Aufruf Oktober 5, 2019).
- [19] API Endpoints Tutorial, YouTube Ethan Jarell. <https://www.youtube.com/watch?v=C470XGASGX0> (Letzter Aufruf August 28, 2019).
- [20] Free Documentation Licence (GNU FDL). <http://www.gnu.org/licenses/fdl.txt> (Letzter Aufruf July 30, 2005).
- [21] Steps toward the Glory of REST. <https://martinfowler.com/articles/richardsonMaturityModel.html> (Letzter Aufruf Juni 30, 2019).
- [22] Hibernate Framework Basic, Java Beginners Tutorial. <https://javabeginnerstutorial.com/hibernate/hibernate-framework-basic/> (Letzter Aufruf Dezember 12, 2019).
- [23] JPA and Hinbernate Tutorial for Beginners with Spring Boot and Spring Data JPA, YouTube-Kanal in28minutes. https://www.youtube.com/watch?v=MaIO_XdpdP8 (Letzter Aufruf November 8, 2019).

- [24] JSP and Servlets Tutorial : First Java Web Application in 25 steps, YouTube-Kanal in28minutes. <https://www.youtube.com/watch?v=Vvnliarkw48> (Letzter Aufruf September 22, 2019).
- [25] Spring framework tutorial for beginners with examples in eclipse | Why Spring Inversion of control. <https://www.youtube.com/watch?v=r2Q0Jz12qMQ> (Letzter Aufruf November 7, 2019).
- [26] JavaDoc, javax.persistence, Annotation @id. <https://docs.oracle.com/javaee/6/api/javax/persistence/Id.html> (Letzter Aufruf Juli 30, 2020). 38
- [27] Javadoc Annotation OneToMany. <https://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html> (Letzter Aufruf August 2, 2020). 39
- [28] Javadoc Klasse Optional. <https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html> (Letzter Aufruf Juli 27, 2020). 41
- [29] Javadoc @PathVariable Annotation. <https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/PathVariable.html> (Letzter Aufruf Juli 30, 2020). 45
- [30] Javadoc PersistenceContext. <https://docs.oracle.com/javaee/7/api/javax/persistence/PersistenceContext.html> (Letzter Aufruf August 4, 2020). 42
- [31] Javadoc Interface Repository. <https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html> (Letzter Aufruf August 2, 2020). 40
- [32] Javadoc, klasse responseentity. <https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html> (Letzter Aufruf Juli 27, 2020). 45
- [33] Javadoc @RestController Annotation. <https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html> (Letzter Aufruf März 4, 2020). 44
- [34] Javadoc Session Interface. <https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html> (Letzter Aufruf August 4, 2020). 42
- [35] Master Hibernate and JPA with Spring Boot in 100 Steps, Udemy Onlinekurs. <https://www.udemy.com/course/hibernate-jpa-tutorial-for-beginners-in-100-steps/> (Letzter Aufruf Januar 15, 2020).
- [36] Oracle Documentation, Primary Key Value Generation. https://docs.oracle.com/cd/E13224_01/wlw/docs103/guide/ejb/entity/conAutomaticPrimaryKeyGeneration.html (Letzter Aufruf Juli 31, 2020). 39
- [37] Pom Reference, Apache Maven Documentation . <https://maven.apache.org/pom.html> (Letzter Aufruf August 27, 2019).
- [38] Spring Rest Docs - Documenting REST API, Example, YouTube-Kanal Java Techie. https://www.youtube.com/watch?v=ghn9p6d__Yc (Letzter Aufruf Februar 3, 2020).
- [39] REST principles explained. <https://www.servage.net/blog/2013/04/08/rest-principles-explained> (Letzter Aufruf Juni 1, 2019).
- [40] Building an Application with Spring Boot. <https://spring.io/guides/gs/spring-boot/> (Letzter Aufruf August 6, 2019).
- [41] How to create a Spring Boot project in Eclipse. <https://www.youtube.com/watch?v=WZzGhWSJ6h0> (Letzter Aufruf Juni 30, 2019).
- [42] Spring Boot API in 10 Minuten - Tutorial Deutsch. <https://www.youtube.com/watch?v=pkVrAmGb1XQ> (Letzter Aufruf August 6, 2019).

- [43] Spring Boot Anwendung mit MySQL Datenbank verbinden - Tutorial Deutsch. <https://www.youtube.com/watch?v=TJfwKT-mx0A> (Letzter Aufruf August 6, 2019).
- [44] Spring Data JPA Reference Documentation. <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods> (Letzter Aufruf Juli 25, 2020). 41
- [45] Java Spring Online Dokumentation. <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html> (Letzter Aufruf September 20, 2019). 27, 44
- [46] Spring Tutorial, JavaTPoint. <https://www.javatpoint.com/spring-tutorial> (Letzter Aufruf Oktober 28, 2019).
- [47] Swagger2 Documentation Tutorial. <https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api> (Letzter Aufruf Januar 5, 2020). 35
- [48] REST API Documentation using Swagger2 in Spring Boot, YouTube-Kanal Tech Primers. <https://www.youtube.com/watch?v=HHyjWc0ASl8> (Letzter Aufruf Februar 3, 2020).
- [49] What is REST API, YouTube-Kanal Telusko. <https://www.youtube.com/watch?v=qVTAB8Z2VmA> (Letzter Aufruf Juni 30, 2019).
- [50] Introduction to Servlets, YouTube-Kanal Telusko. <https://www.youtube.com/watch?v=CRvcm7GKrF0> (Letzter Aufruf Sept 19, 2019).
- [51] What is Spring Boot? Introduction, YouTube-Kanal Telusko. <https://www.youtube.com/watch?v=Ch163VfHtvA> (Letzter Aufruf September 23, 2019).
- [52] What is a RESTful API? Explanation of REST and HTTP, YouTube-Kanal Traversy Media. <https://docs.spring.io/spring/docs/2.0.x/spring-reference.pdf> (Letzter Aufruf August 28, 2019).
- [53] Domain Model, Wikipeda. https://en.wikipedia.org/wiki/Domain_model (Letzter Aufruf Februar 5, 2020).
- [54] JavaBeans, Wikipedia. <https://de.wikipedia.org/wiki/JavaBeans> (Letzter Aufruf Februar 1, 2020).
- [55] Apache Maven, Wikipedia. https://de.wikipedia.org/wiki/Apache_Maven (Letzter Aufruf August 27, 2019).