,

REST API mit Java Spring

Speicherung und Zugriff auf Statistiken einer just-for-fun Eishockeymannschaft mit Spring Boot, Hibernate und Postgres

Bachelorarbeit

Marc Raemy

Februar 2020

Unter der Aufsicht von:

Prof. Dr. Jacques Pasquier-Rocha
and
Pascal Gremaud
Software Engineering Group

UNI FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Software Engineering Gruppe
Department für Informatik
Universität Freiburg (Schweiz)

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, die mich sonst in irgendeiner Form bei der Arbeit unterstützt haben.

Abstract

In dieser Arbeit wurde ein Application Programming Interface (API) mit dem Java Spring Framework, Hibernate und einer Postgres-Datenbank nach REST-Prinzipien erstellt und dokumentiert. Über das Interface können Statistiken für eine just-for-fun Eishockeymannschaft gespeichert und gelesen werden. Ziel der Arbeit war, Spring, Spring Boot, Hibernate und REST anzuwenden und darzustellen.

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

1.1. Motivation und Ziel

Da der Autor mit Freunden regelmässig Eishockey spielt und in den Spielen Statistiken geführt werden (über Gegner, Resultat, anwesende Spieler, Torschützen und Assistgeber), entschied er sich als Thema für die Bachelorarbeit, einen Webservice nach den REST-Prinzipien zu programmieren, mit dem die Statistiken online abgerufen und verwaltet werden können. Die Skala der Arbeit konzentriert sich auf die Entwicklung der Serverseite und die Dokumentation der API mit einem der gängigen Dokumentationstools wie z.B. Swagger. Diese sollte ausserhalb des Rahmens dieser Arbeit um einen Client erweitert werden können.

Als Technologien wurden das Java Spring Framework und Spring Boot, sowie objekt/relationales Mapping (ORM) mit Hibernate gewählt. Das Ziel der Arbeit ist, die aktuellen Java-Tools für Webentwicklung sprich das Spring-Framework kennen zu lernen, REST anzuwenden und die API für die Eishockeymannschaft zu erstellen.

1.2. Aufbau der Arbeit

Einleitung

In der Einleitung sind die Ziele der Arbeit, ein Überblick über die Struktur der Arbeit sowie Informationen zu Notation und Konventionen dargestellt.

Kapitel 1: Verwendete Frameworks und Konzepte: Spring Boot, Hibernate/ORM und REST

In diesem Kapitel werden die in dieser Arbeit verwendeten Technologien, das Spring Framwork, Spring Boot, Hibernate sowie die REST-Prinzipien, vorgestellt und soweit wie möglich erläutert.

Kapitel 2: Datenmodell und Endpoints

In diesem Kapitel werden die Daten, das Entitäten-Beziehungs-Modell und die daraus resultierenden Tabellen dargestellt. Daraus werden die für die Funktionalitäten und Ressourcen benötigten Endpoints abgeleitet und beschrieben.

Kapitel 3: Programmierung, Quellcode und Dokumentation der API

In diesem Kapitel werden die Struktur des Quellcodes, der Quellcode, das Vorgehen bei der Programmierung und Dokumentatin der API sowie die wichtigsten Klassen und Interfaces dargestellt.

Chapter 4: Abschliessende Bemerkungen

Hier wird die ursprüngliche Idee der Arbeit noch einmal beschrieben und ob die Ziele der Arbeit erreicht worden sind. Weiter werden die Probleme und Herausforderungen während der Durchführung der Arbeit sowie deren Nutzen beschrieben.

Anhang

Enthält Literatur- und Onlinereferenzen, die Liste der Abkürzungen sowie Ausschnitte aus Artefakten, die für diese Arbeit genutzt wurden,

1.3. Notation and Konventionen

- Formatierung:
  - Abkürzungen werden wie folgt verwendet: Java Persistence API (JPA) bei der ersten Verwendung und JPA 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 vier 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 *Figure i.j.*
- Bezüglich der Verwendung der geschlechtlichen Form von Nomen wird die Konvention mit weiblicher Form und Stern verwendet, z.B. Entwickler*innen. Männer und intersexuelle Personen sind in dieser Form ebenfalls eingeschlossen.

2

Frameworks und Konzepte: Spring, Spring Boot, Hibernate und REST

2.1. Spring Framework

Die erste Version des open source-Frameworks Spring wurde vom australischen Programmierer und Musikwisenschaftler Rod Johnson 2002 im Buch „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. Diese sind mit anspruchsvollen Anforderungen wie komplexe Datenbanktransaktionen, Authentifizierung, Sicherheit, Zugang übers Web oder Monitoringsfunktionalitäten konfrontiert. [8] Er suchte nach einem Weg, Businessapplikationen auf möglichst einfache und schnelle Weise programmieren zu können.

2.1.1. Vorläufer: JavaBeans

In den Neunzigerjahren wurde von Sun Microsystems die JavaBeans-Spezifikation entwickelt [42]. Dabei handelt es sich um ein Softwarekomponentenmodell, das Entwicklungsrichtlien enthält, die die Wiederverwendung von einfachen Java-Objekten und ihre Zusammenstellung zu komplexeren Applikationen vereinfachen sollen [8, S.4]. Sie entstanden aus der Notwendigkeit heraus, Java-Klassen möglichst einfach 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) [42]

JavaBeans wurden ursprünglich in erster Linie für die Entwicklung von Benutzeroberflächen-Widgets eingesetzt. Für die höhere Anforderungen stellende Businessapplikationen erschienen sie als nicht geeignet. [8].

Enterprise Java Beans (EJB)

Ein erster Versuch JavaBeans im Businesskontext einzusetzen waren EJB. Damit sollten komplexere Anwendungen für Bedürfnisse von Unternehmen mit Einsatz der flexiblen und leichtgewichtigen JavaBeans entwickelt werden. Die Kombination dieser Konzepte in EJB war jedoch zu schwerfällig und komplex, weshalb sich die Entwickler davon abwandten. [8, S.4].

Parallel dazu wurde das Spring Framework entwickelt, mit dem Ziel, die komplexeren Anforderungen an Unternehmensapplikationen mit gleichzeitiger Verwendung von JavaBeans erfüllen zu können. [8]. Es bietet eine Implementation formalisierter best practices der Softwareentwicklung im Business-Kontext und somit einen Vorschlag für eine Architektur, die sicher, stabil und für einen einfachen Unterhalt der Applikationen geeignet ist. Dies unter Anderem indem es die Konzepte der Dependency Injection (DI) und des Aspect Oriented Programming (AOP) einsetzt [4].

Spring ist das bekannteste und am weitesten entwickelte Framework für die Entwicklung mit JavaBeans. [8, S.4]. Der Begriff "Bean" oder "JavaBean" wird dabei relativ locker verwendet. Er bezeichnet oft einfach eine Anwendungskomponente, eine Instanz einer Klasse, ohne in jedem Fall der JavaBeans-Spezifikation [2] bis ins letzte Detail zu folgen. [8, S.5].

Die wichtigsten Funktionen des Spring Frameworks sind, loose coupling, statt tight coupling zu ermöglichen, dieses zentrale Konzept wird durch "dependecy injecton" umgesetzt. In einer grösseren Applikation werden eine grosse Anzahl Klassen instantiiert (diese Instanzen werden "Beans" genannt). Diese Klassen hängen von anderen Objekten ab, die sie als Felder oder für Ihre Methoden benötigen, ihre dependecies, (in der Java-Sprache wird das Instantiieren von Klassen, die von einer anderen Klasse dann genutzt werden, dependencies, "wireing" genannt. Die Instanzen, d.h. dependencies, werden mit den sie verwendenden Beans "gewired"). Die Instantiierung wird dabei eben nicht hartcodiert im Quellcode implementiert, sondern durch das Framework via Konfigurationsdatei der Applikation. Die Konfiguration teilt dem Framework mit, welche Objekte (Beans) instantiiert werden sollen und von welchen anderen Objekten diese genutzt werden sollen. [26]

Dependencies

Dependencies sind Libraries, Dokumente, Teile anderer Programme oder des Codes derselben Applikaiton, die ein Programm braucht, um korrekt ausgeführt werden zu können. Built tools helfen, diese Links herzustellen, so sind alle Ressourcen verfügbar, die nötig sind, damit die Applikation ausführbar wird. Innerhalb einer Klasse werden andere Klassen, welche diese Klasse benötigt, als Dependencies bezeichnet. Die Klasse ist dann abhängig von Instanzen dieser übrigen Klassen, um ihre Funktionalität auszuführen.

Dependency injection

Es geht vor allem darum, möglichst wenig fixe, hartcodierte Abhängigkeiten zwischen einzelnen Klassen zu kreieren (loose coupeling statt tight coupeling), somit kann die Applikation besser gewartet, adaptiert und migriert werden. [35]

Dependency Injection bedeuted, dass die Objekte nicht im code selber instantiiert werden, sondern das Framework instantiiert die Objekte zur Laufzeit (in der Java Welt werden diese Instanzen, die das Rückgrad der Applikation bilden und vom Container gemanagt werden, "Beans" genannt). Der für die Dependecy Injection zuständige "Container", liest die Konfigurations-Metadaten und "injeziert" die Instanzen und ihre Dependencies in den Kontext. Das Programm wird so konfigurierbar, ohne den Code selber zu verändern. Der Container in Spring ist eine Implementation eines Interfaces mit dem Name "Application Context". [24]

Die Metadaten können in xml, als Java Annotationen oder als Java Code definiert sein. [Spring Reference Dokumentation 5.2.0] Beans sind die Instanzen der Objekte, die vom Framework zur Laufzeit erstellt werden und von diesem gemanagt werden. Eine sogenannte Bean Factory, die dem Factory-Pattern folgt, erstellt, unter Einbezug der Konfigurationen aus einer xml-Konfigurationsdatei, Instanzen und übergibt sie an die aufrufenden Klassen. Beans und die dependencies zwischen ihnen sind in den Konfigurations-Metadaten abgebildet, auf die der Container, der die Beans managt, zurückgreift. [Quelle: Spring 5.2.0 Reference Dokumentation]

Aspect Oriented Programming

Nebst der dependency injection (DI) ist die Aspektorientierte Programmierung das zweite Kernfeature von Spring. [8] Aspects sind nichts anderes als spezifische Java Klassen, die Anforderungen (sogenannte "concerns" in der AOP-Terminologie) enkapsulieren und implementieren, welche über verschiedene Schichten und Klassen der eigentlichen Applikation benötigt werden, wie eben z.B. Sicherheit, Authentifizierung, Transaktionen. Sie werden so losgelöst von der Funktionalität der anderen Programmteile, und flexibel mit ihnen verschaltet. Dadurch erspart man sicher z.B. dass man in jeder Klasse einer Applikation eigenen z.B. Sicherheits-Code implementieren muss, was zu unwartbaren und mit viel sich wiederholendem Code ausgestatteten Applikationen führt. [8, S.5]

Servlet

Servlet steht für "Server Component". Allgemein steht das Postfix "let" in der Java-Welt für "Component". Es wird z.B. auch verwendet für "Applet" (= Component einer Applikation) etc.. Bei einem Request für eine dynamische Webseite ist auf Server-Seite das Servlet dafür zuständig, die entsprechende Response zu liefern. Es ist nichts anderes als eine Java-Klasse, die einen HTTP-Request entgegen nimmt und sich darum kümmert, die entsprechende Antwort zu liefern. [23]

2.1.2. Spring Module

Das Spring Framework setzt sich aus einer grossen Anzahl von Modulen zusammen, die für eine grosse Vielfalt von Funktionalitäten und Applikationen Libraries und Frameworks

zur Verfügung stellen. Diese können online mit dem Spring Initializr, https://start.spring.io/ durch Anwählen von Checkboxes je nach Bedarf ausgewählt werden. Spring ist in diesem Sinne nicht nur ein Framework, sondern ein Framework of Frameworks.[8]

Spring Framework Core

Basis von allem ist das core Spring Framework. Es stellt den core-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 es weitere essentielle Komponenten wie das Spring MVC Framework, welches das Hauptframework für Webapplikationen ist, sowie Support für elementare Datenspeicherungsfunktionen in einer Datenbank mit Java Database Connectivity (JDBC).[9, S.26].

Spring Boot

Spring Boot ist mittlerweile essenzienter Bestandteil der Entwicklung mit Spring. Es ermöglicht eine sehr schnelle und einfache Entwicklung von Applikationen, indem Autokonfiguration der Applikationen zur Verfügung stellt.[9] Siehe dazu Abschnitt 2.1.3..

Spring Data

Über die Basisunterstützung für Datenbankkonnektivität des core-Frameworks hinaus bieten die Datenbank-Module von Spring Unterstützung für die Verbindung mit nahezu allen gängigen Datenbanktechnologien. Dazu bietet Spring eine sehr einfache Art, diese Datenbankkonnektivität zu implementieren, indem einzig ein simple Java-Interfaces definiert werden müssen [9].

Spring Security

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

Spring Integration und Spring Batch

[9].

Spring Cloud

[9].

Unterscheidung Spring Framework, Spring MVC and Spring Boot

Spring MVC ist selber ein Framework für Webapplikationen. Es ist Teil von Spring. Spring Boot ist ein Tool, um möglichst schnell und ohne viel Programmieraufwand, der nicht zur Logik der Applikation gehört, eine automatisch konfigurierte, lauffähige Applikation mit Spring zu erstellen [9, S. 25]

Ein wichtiges Problem, um welches sich das Spring Framework kümmert, ist tight coupling. Es übernimmt die dependency injection und implementiert das inversion of control-Prinzip. Das Framework managt und erstellt die Instanzen der Klassen (Beans) und die Übergabe derselben an andere Klassen (depenencies). Somit ermöglicht es lose gekoppelte Applikationen. Diese können z.B. viel einfacher getestet werden, als wenn die Instanzen hard codiert in den Klassen erstellt werden [26] Dazu erspart es das Schreiben von Boilerplate-Code.

Spring stellt mit Spring MVC ein Framework für Webapplikationen zur Verfügung. [10]. Spring MVC implementiert dabei das Model-View-Controller (MVC)-Schema. Mit Model wird die Ressource oder die Daten bezeichnet, welche der Server auf eine Web-Anfrage hin übermittelt. Die View bezeichnet eine geeignetes Format, eine geeignete Darstellung dieser Daten, der den Anforderungen des Clients, der eine Anfrage übers web sendet, entspricht und der controller nimmt die Anfrage entgegen und kümmert sich um die Weiterleitung an entsprechende Programmteile, die die Anforderungen der Anfrage bedienen können. [8]

2.1.3. Spring Boot

Autokonfiguration

Während das Spring-Ökosystem eine breite Palette von Frameworks für Funktionalitäten von Business-Applikationen abdeckt, konzentriert sich Spring Boot darauf, mit möglichst wenig Code und schnell das Erstellen einer kompletten, einsatzfähigen Applikation, insbesondere von Microservices [26] zu ermöglichen. [17] Dies geschieht, indem sich Spring Boot um die Konfiguration der Applikation kümmert, gemäss best Practices und Konventionen, die sich in der Industrie etabliert haben. Es setzt damit den Grundsatz "Convention over Configuration" um. [39]

Die Autokonfiguration ist ein zentrales Merkmal und ein zentraler Nutzen von Spring Boot. Es prüft die Klassen und JAR-files im classpath und konfiguriert basierend darauf automatisch die Applikation. [22] Wenn z.B. Hibernate im classpath ist sowie eine spezifische Datenbank, konfiguriert Spring Boot Hibernate für diese spezifische Datenbank. Gleichzeitig ist es auch das Ziel, trotzdem einfach manuelle Änderungen an der Konfiguration vornehmen zu können.

Metrics, Testing und Spezifikation der Umgebung

Nebst der Autokonfiguration sind weitere Ziele von Spring Boot, der Applikation häufig benötigte, nichtfunktionale Featues zur Verfügung zu stellen. Insbesondere eingebettete Server, Überwachung (metrics) und health checks für die Applikation. Die entsprechende Library für Metrics heisst Spring Boot Actuator. Damit kann man z.B. prüfen, ob ein Service erreichbar ist, wie oft er aufgerufen wurde, wie oft er fehlgeschlagen ist etc.. Weiter stellt Spring Boot zusätzliche Unterstützung für das Testen der Applikation zur Verfügung im Vergleich zum core Framework. Dazu ermöglicht es das flexible Spezifizieren von Umgebungseigenschaften der Applikation [9].

Spring Boot generiert keinen Code und ist kein Applikations- oder Webserver [26]. Es soll es der Prgrammier*in ermöglichen, sich auf die Logik und die Funktionalität der

spezifischen 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 Tomcat Server ein Teil einer Spring Boot Applikation und wird automatisch für diese konfiguriert.

Das Konzept der 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, der auf der entsprechenden Maschine läuft. Für Microservices, die über ein Netzwerk verknüpft sind, bietet das eine grosse Vereinfachung. [26]

2.2. ORM und Hibernate

2.2.1. Objekt/relationales Mapping

Object/relational Mismatch

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 Paxis über Jahre hinweg entwickelt wurde [1]. Hibernate ist eines der ersten und bekanntesten ORM-Frameworks. Bei früheren Datenbank-Mapping-Tools oder Libraries wie JDBC musste man teilweise aufwändigen Code schreiben, die Queries von Hand ausformulieren und die Ergebnisse von Hand weiterverarbeiten. Mit Hibernate und ORM muss man Queries nicht mehr selber schreiben. Das Problem beim Queries selber Schreiben ist, dass wenn die Datenbank verändert wird, auch die Queries angepasst werden müssen. Das kann eine sehr aufwändige Aufgabe sein, wenn man viele und grosse Queries für eine grosse Applikation verwalten muss. [22]

JDBC und Spring JDBC

Java Database Connection (JDBC) ist die Library in Java welche Abfragen auf Datenbanken ermöglicht. Dabei müssen jeweils die Verbindugen und das Trennen von Verbindungen zur Datenbank gemanagt werden, sowie das SQL-Statement, das Resultat muss von Hand iteriert und die Ergebnisse einzeln ausgelesen werden, diese müssen gespeichert und zurückgegeben werden und alle möglichen Arten von Fehlern müssen behandelt werden. Kurz: es braucht viel Code und es ist aufwändig, sich um alles kümmern zu müssen.

Spring JDBC vereinfacht diesen Prozess merklich, in dem es Automatisierungen vieler dieser Schritte zur Verfügung stellt und Programmierung auf abstrahierter Ebene ermöglicht, die viel weniger Code und Aufwand benötigt. [26]

Deutlich mehr Features und leistungsfähiger als JDBC und Spring JDBC sind Tools und Framewokrs für objekt/relationales Mapping. Vor allem, wenn Anwendungen komplexer werden, reichen JDBC-Tools oft nicht mehr aus. Hierbei werden mit [8, S. 34]

In JDBC schreibt queries, ordnet ihnen Werte aus der Java Applikation zu, die man den Tabellen der Datenbank zuordnet (mapping). Oder ordnet mit Queries Werte aus der Datenbank den Java-Objekten zu. Man schreibt die Queries von Hand und kümmert sich von Hand um das korrekte Mapping. Das geht gut, solange es sich um simple Queries und kleine Applikationen handelt. In grösseren Applikationen bestehen jedoch hunderte von Tabellen und die Queries sind hunderte Zeile Code gross. Somit wird die Wartung und Änderung der Applikation zu einer extrem aufwändigen und schwierigen Aufgabe mit JDBC.

Mit objektrelationalem Mapping kann diese Aufgabe viel besser bewältigt werden. Die Java Persistence API und Hibernate (JPA) sind bekannte Open Source ORM-Frameworks. Sie definieren Sets von Annotationen und Interfaces, mit denen das Framework automatisch die Queries und Tabellen der Datenbanken erstellt. [26]

Spring Data JPA (Java Persistence API) ist das Persistenzmodul von Spring, das den Support für die Verbindung zu und das Ablegen, Lesen und Ändern von Daten in Datenbanken und der Modifikation von Datenbanken selber bereitstellt. Es bietet auch Support für objekrelationales Mapping., das verbreitetste ORM-Framework "Hibernate" implementiert JPA, bietet aber noch mehr Funktionalitäten. Das Problem ist, dass, wenn man für jede Entität eine Klasse schreiben müsste, einen Eintity Manager, mit der man dieses Objekt speichern könnte, müsste man die gleichen methoden (save, get, insert, delete, merge, find..) jeweils immer neu schreiben. Spring Data JPA ist ein Modul des Springframeworks, welches diese Funktionalitäten in abstrahierter Form zur Verfügung stellt. Man muss nur ein Interface für jede Entität definieren, Spring Data JPA implementiert dann dieses Interface automatisch. Das Interface JpaRepository stellt eine von mehreren Abstraktionen zur Verfügung für diese Datenankfunktionalitäten. [22]

Um ORM umzusetzen muss man angeben, wie die Klassen mit den Datenbanktabellen in Verbindung stehen. Wenn die entsprechenden Annotationen in den Klassen gesetzt sind, kann Hibernate diese lesen, interpretieren und die entsprechenden Queries in den Datenbanken vornehmen [26]. Sie ersetzen also das oft aufwändige von Hand Programmieren des Zugangs und der Abfragen auf die Datenbank durch die möglichkeit, durch wenige Annotationen in den Java Klassen all diese funktionalitäten automatisch zu generieren. [21]

Damit Hibernate die korrekten Abfragen generieren kann, und da jede Datenbanktechnologie einen leicht unterschiedlichen sql-dialekt implementiert, muss der sql dialekt, respektive die Datenbank in Spring Boot angegeben werden werden. Dies wird in einer Datei spring.jpa.properties.hibernate.dialect, gespeichert, die via Konfigurationsdatei ("application properties") des Maven/Spring Boot-Projekts eingelesen wird. Hibernate kann so flexibel mit unterschiedlichen Datenbank-Technologien mit sehr geringem Konfigurationsaufwand verbunden werden. https://javabeginnerstutorial.com/hibernate/hibernate-framework-basic/, 25.11.2019

Transaktionen: JPA Persistence Context respektive Hibernate Session

Aus den Enterprise Java Beans Spezifikationen stammt das Konzept der Entity-Beans. Das sind Business-Objekte, die in einer relationalen Datenbank persistent gehalten werden. Der Trend der Java Entwicklung ging weg von den komplexeren EJB-Objekten zu simpleren Plain Old Java Objects (POJO's) die auch für Persistenz-Funktionalitäten verwendet werden können, was zuvor noch nicht der Fall war.

Die Java-Persistence API (JPA) entstand aus den EJB-Entity-Beans und dem POJO-Trend als darauffolgender Persistenz-Standard, der sich unter anderem auch von Hibernate inspierieren liess. Spring unterstützt JPA und die Verwendung von JPA mit Spring wurde fortan, das war im Jahr 2012, eine weit verbreitete Methode zur Persistenzimplementation mit Java. [Walls, 2012, S.148].

Spring unterstützt die deklarative Transaktionsverwaltung mit Hilfe des Spring Aspect Oriented Programming (AOP)-Frameworks. Transaktionen werden als Aspekte, d.h. Dienste, die über verschiedene Programmteile hinweg benötigt werden, implementiert. [Walls, 2012, S.165]. Dass Spring Unterstützung für deklarative Transaktionsverwaltung für ganz normale Java Objekte ("Plain old Java Objects", POJO's), ist ein wesentliches Feature von Spring. [Walls 2012, S.165]

JPA EntityManager ist ein Interface zu einem sogenannten Persistence context oder einer Session, wie dies in Hibernate genannt wird. Dieses Interface stellt die grundlegenden Methoden für die Datenmanipulationen, z.B. Daten aus der Datenbank lesen, Daten in der Datenbank speichern, löschen oder updaten zur Verfügung. Wann immer wir einen EntityManager benutzen, interagiren wir mit dem Persistence Context. Er bezeichnet das Set der Objekte, von denen Informationen in die Datenbank übertragen werden sollen. Mit der Annotation @Transactional bei einer Methode, kann man signalisieren, dass es sich um eine Transaktion handelt, dabei wird der Inhalt der Transaktion nur als Ganzes, erst am Schluss, wenn alle Instruktionen in der Methdode erfolgreich ausgeführt werden konnten, in die Datenbank übertragen und die Queries ausgeführt. Wenn es Fehler gab, werden die vorherigen Operationen der Transaktion annuliert mit einem Rollback rückgängig gemacht. [in28Minutes, 2019]. Spring Boot und Spring verwalten jedoch die Transaktionen zu einem grossen Teil selbsttändig, so dass man nicht von Hand die Transaktionen angeben muss. [Simons, 2018, S.213]

Der Persistence context respektive die Session ist der Ort, wo alle Änderungen aufgezeichnet, vermerkt und verfolgt werden. Die Session wird am Anfang der Transaktion (normalerweise einem Methodenaufruf) kreiert und nach dem Ende aller Operationen. aufgelöst Wenn die Annotation @Transactional nicht gesetzt wird, kann in gewissen Fällen jede einzelne Operation, jeder Methodenaufruf, als eigene Transaktion behandelt und direkt in die Datenbank gespeichert werden. Das heisst, dass der Persistence Context nach jeder Instruktion wieder aufgelöst wird. Wenn eine nachfolgende Operation auf eine vorher verwendetes Objekt, auf das eine Datenbankoperation ausgeführt wurde, zugreifen will, wird eine Fehlermeldung, z.B. "Lazy Initialisation Exception - No Session", ausgelöst, weil keine Session mehr vorhanden ist und somit das Objekt nicht mehr existiert. [26]

Der Persistence Context ist ein wichtiges Konzept von JPA respektive die Session ein wichtiges Konzept von Hibernate. Es ermöglicht die Verbindung zur Datenbank und bewirkt, dass die Änderungen an den Daten, die mittels EntityManager gemacht werden, registriert und in der Datenbank abgespeichert werden. Die Klasse, die die Zugriffe implementieren werden Data Access Objects (DAO's) genannt. In den letzten Jahren vermehrt "Repository". Sie werden eingesetzt um auf die Session zuzugreifen und die Datenoperation vorzunehmen.

In Hibernate 5.4 müssen jedoch die Operationen im Data Access Object nicht mehr definiert werden und kein EntityManager mehr von Hand eingesetzt werden. Wenn die dependency spring-boot-starter-jpa es muss lediglich ein Interface erstellt werden werden, welches das "Repository"-Interface, das das Haupt Marker Interface für Operationen auf eine Datenbank, respektive eines seiner Sub-Interfaces, z.B. das CRUD-Repository,

erweitert werden. Sie stellen die Methodendefinition für die Datenmanipulation zur Verfügung. Es muss bei der Erweiterung lediglich der generische Datentyp, die Klasse, deklariert werden, die dem Typ der Daten respektive der Struktur der Tabelle entspricht, die in der Datenbank gespeichert oder aus ihr gelesen werden sollen. Man muss selber keine Implementierung der Zugriffsmethoden schreiben, es findet auch keine Codegenerierung oder Bytecode-Manipulation statt. Spring Data generiert zur Laufzeit dynamische JDK-Proxys. Die Methodenaufrufe werden mit einer MethodInterceptor-Klasse abgefangen, die zuerst schaut, ob es eine konkrete Implementierung des Repositorys gibt, falls nicht, versucht Spring Data, die SQL-Abfrage zu ermitteln, z.T. sind Abfragen im Code definiert, ansonsten über den Methodennamen und falls das nicht gelingt, greift Spring Data auf eine Basisklasse zurück, in der die Methoden implementiert sein müssen. [Simons, 2018, S. 221] [in28Minutes, 2019]

2.3. REST Prinzipien

REST API's sind ein leichtgewichtiger und weborientierter Ansatz für Services. [Pasquier, 2019]. Es ist der Ansatz der in den letzten Jahren am populärsten geworden ist und SOAP den Rang abgelaufen hat. Es handelt sich bei REST (Representational State Transfer) um ein Set von Richtlinien und Prinzipien (somit um ein Entwurfsmuster oder ein Architekturstil) für die Programmierung von Schnittstellen (Interfaces) zwischen verteilten Systemen und verteilten Applikationen. [Tilkov et al., 2015, S. ix] REST basiert auf HTTP. Die allgemeinen Ziele für verteilte Systeme, die REST zu erreichen hilft, sind loose Kopplung, Interoperabilität, Wiederverwendung, Performance und Skalierbarkeit. [Tilkov et al., 2015, S. 3f]. REST-Abfragen sind schneller und datenärmer als SOAP-Abfragen, da ein weniger grosser Overhead mitgeschickt wird als beim zuvor am weitesten verbreiteten Standard "Simple Object Acess Protocol (SOAP). Im Unterschied zu SOAP-Webservices funktioniert die Abfrage bei REST-API's nur über die URL.

REST Prinzipien Nutzung von Hypermedia ..

3
Datenmodell und Endpoints

3.1. Rohdaten

In dieser Arbeit wird ein Restful Web-Service mit dem Java Spring Boot Framework erstellt, mit dem Statistiken einer Eishockeymannschaft zur Verfügung gestellt werden. Es sind Statistiken einer realen Freizeit-Eishockeymannschaft aus Freiburg, dem "HC Keile".

Die Daten stehen aktuell als Excel-Auszug in einem txt Rohdatensatz zur Verfügung. Sie enthalten Informationen zu: Namen der Spieler, die ein spezifisches Spiel gespielt haben, Anzahl Tore, erste und zweite Assists aller Spieler, auf welcher Position die Spieler gespielt haben (G=Goalie, S=Sturm, C=Center,V=Verteidiger), wie viele Gegentore der Torhüter hinnehmen musste, ob das Spiel gewonnen, verloren, unentschieden gespielt wurde oder intern war (intern = beide Mannschaften bestehen aus Spielern des HC Keile), sowie in welcher Saison das Spiel stattgefunden hat. Die Einträge sind nummeriert von 1 bis 1562, zum Teil hat das txt file Formatierungsfehler.

3.1.1. Auszug aus Rohdatensatz Statistiken HC Keile

ID MatSaison MachNr 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 1 2 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 1 0 0 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 S 2 0 0 0 verloren 1164 201718 1 Flogge C 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 ... (MatSaison = Saison, MachNr = Matchnummer, SpAka = Name des Spielers, SpPos = Position, SpT = erzielte Tore, SpA1 = Anzahl erste Assists, SpA2 = Anzhal zweite Assists, GeT = Gegentore (nur für Goalie), SpRes = Resultat).

Zusätzlich zu diesen Daten bestehen auf der Facebook Seite, im whatsapp-chat der Mannschaft, auf doodle sowie auf privaten Rechnern weitere Daten wie Informationen zum Name des jeweiligen Gegners und kurze Spielberichte. Weiter werden innerhalb der Mannschaft Preise für besondere Aktionen verliehen und es gibt ein Spiel, bei dem der Gewinner einen Pokal erhält. Diese Daten könnten später - nicht im Rahmen dieser Arbeit - ebenfalls im Webservice aufgenommen werden.

Für diese Arbeit ist es jedoch das Ziel, die wesentlichen Prinzipien einer Restful-API zu verstehen und sauber zu implementieren. Daher konzentriert sich die Arbeit auf die Scoring-Daten zu den Spielen, und darauf, diese sauber zu implementieren, mit dem Hauptziel, die Basiskonzepte eines Restful Webservice zu verstehen.

3.2. Entitäten-Beziehungsmodell und Datenbankschema

3.2.1. Ressourcen, die vom Webservice zur Verfügung gestellt werden

Die Nutzenden des Webservices sollen einen Spieler auswählen und Informationen erhalten zu: Wie viele Spiele er gespielt, wie viele Tore er erzielt und wie viele erste und zweite Assits er produziert hat. Zweitens sollen die User ein Spiel auswählen und Daten zum Resultat, welche Spieler gespielt haben, wie viele Tore und Assists jeder erzielt hat, erhalten.

Die Daten können in folgendem Entitäten-Beziehungsmodell dargestellt werden:

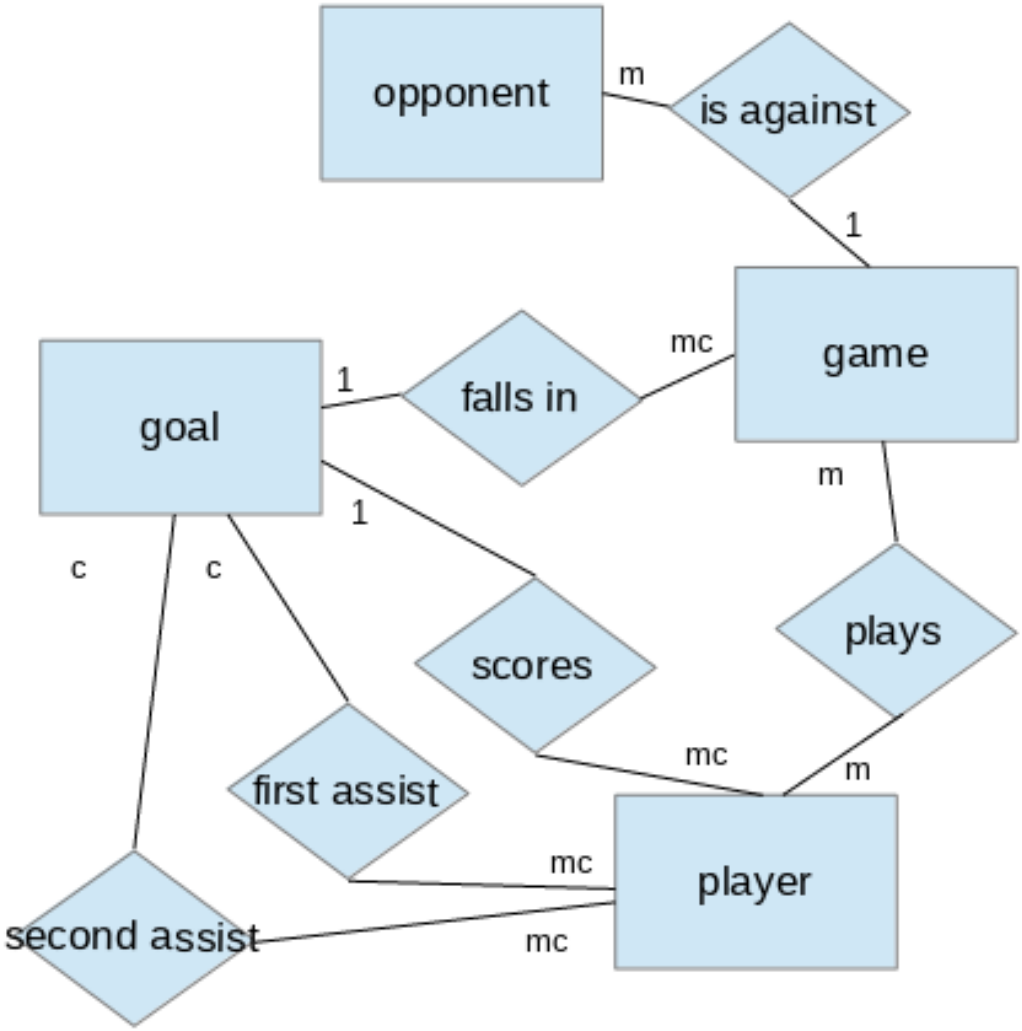

Abbildung 3.1.: Entitäten-Beziehungsmodell

3.2.2. Verbale Beschreibung

Mehrere Spieler (player) spielen ein Spiel (game). Ein Spiel hat keins, eins oder mehrere Tore des HC Keile (goal). Einem Spiel ist genau eine gegnerische Mannschaft zugeordnet. Diese kann dies in einem oder mehreren Spielen sein.

Ein Spieler spielt eins oder mehrere Spiele. Er kann keine, eins oder mehrere Tore, erste (first assist) oder zweite (second assist) Assits erzielt haben. Die Assists sind als Attribut einem bestimmten Tor zugeordnet, das wiederum einem Spiel zugeordnet ist. Im Fall des Torhüters ist für jedes Spiel vermerkt, ob er keins, eins oder mehrere Gegentore erhalten hat. Eigentore gibt es im Eishockey nicht, wenn eine Mannschaft den Puck ins eigene Tor manövriert, gilt der Spieler der gegnerischen Mannschaft, der den Puck als letzter berührt hat, als Torschütze.

Ein Tor wird genau einem Spiel zugeordnet. Es enthält neben der Spieler-ID des Torschützen die ID des ersten und zweiten Assistgebers als Attribute. Ein Tor kann keins, eins oder zwei Assists haben. Jedes erste Assist und jedes zweite Assist gehören zu genau einem Tor. Jedes Tor, jedes erste Assist und jedes zweite Assist sind genau einem

Spieler zugeordnet. Zwischen den Entitäten “player” und “game” besteht die komplex-komplexe Beziehung “plays” (Spieler spielt Spiel). Diese Relation wird in einer eigenen Tabelle abgebildet.

3.2.3. Tabellen

Daraus ergeben sich die folgenden Tabellen: Je eine Tabelle pro Entität: “game”, “goals”, “opponent” und “player”, dazu eine Tabelle für die komplex-komplexe Beziehung “player plays game”. Die restlichen Relationen werden mittels eines Fremdschlüssels in den erwähnten Tabellen modelliert.

attribute	format	primary key	foreign key	not null
game_id	INT	✓		✓
game_date	DATE			✓
game_status	CHAR			
goals_opponent	INT			✓
goals_keile	INT			
opponent_id	INT		✓	

Tabelle 3.1.: Tabelle Game

attribute	format	primary key	foreign key	not null
goal_id	INT	✓		✓
game_id	INT		✓	✓
scorer_id	INT		✓	✓
assist1_id	INT		✓	
assist2_id	INT		✓	

Tabelle 3.2.: Tabelle Goal

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

Tabelle 3.3.: Tabelle Player

[redacted]	format	primary key	foreign key	not null
opponent_id	INT			✓
name	CHAR			

Tabelle 3.4.: Tabelle Opponent

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

Tabelle 3.5.: Tabelle Games played

3.3. Anforderungen und Endpoints

3.3.1. Anforderungen

Der Webservice soll die Statistiken zu den Spielen und Spielern zur Verfügung stellen, zweitens sollen Daten von Spielen und Spielern via den Web Service in eine Datenbank eingetragen werden können. Somit ergeben sich folgende Use Cases für einen Client:

- 1.a Statistiken der Spieler konsultieren

Für die einzelnen Spieler sollen Name, Vorname, Anzahl gespielter Spiele, Anzahl geschossene Tore, Anzahl erste Assists, Anzahl zweite Assists sowie Summe davon insgesamt und im Durchschnitt pro Spiel konsultiert werden können. An der jährlichen Generalversammlung präsentiert der Statistikzuständige des HC Keile jeweils die Statistiken der abgelaufene Saison sowie insgesamt über alle Saisons. Als eine Variante könnten die Spieler, die in der Datenbank erfasst sind, dem user auf dem Client angezeigt werden. Er könnte dann einen Spieler auswählen, dessen Statistiken im Detail angezeigt werden.

Als zweite Variante könnte eine Liste mit den Statistiken von allen Spielern direkt angezeigt werden. Diese könnten dann nach gewünschten Parametern wie Saison oder Position des Spielers sortiert werden. Das wäre analog zum Beispiel der Statistik-Seite das Schweizerischen Eishockeyverbandes SIHF : https://www.sihf.ch/de/game-center/national-league/#/mashup/players/player/points/desc/page/1/

- 1.b Statistiken der Spiele konsultieren

Zweitens sollen Daten zu Spielen konsultiert werden können: Datum des Spiels, Saison, Resultat, Name, Vorname der Spieler des HC Keile, die das gespielt haben, Tore, Assists der Spieler.

Diese Zahlen werden ebenfalls an der Generalversammlung präsentiert. Es könnte z.B. eine Liste mit allen Spielen im Client angezeigt werden (Datum, Gegner, Resultat). Auf Auswahl könnten dann auch die Daten der Spieler aus den einzelnen Spielen angezeigt werden.

- 2. Spielstatistiken in Datenbank übermitteln

Bei den Spielen des HC Keile werden die Statistiken jeweils von Hand notiert: Resultat, anwesende Spieler, Torschützen und Assistgeber für die Tore. Die Daten werden dann ein zweites Mal ebenfalls von Hand in ein Excel-File übertragen, das auf einem privaten Rechner abgespeichert ist und das nicht online zugänglich ist.

Der Web-Service soll die Funktionalität bereit stellen, dass man die Daten am Spiel direkt via Web-Formular in eine Online-Datenbank übertragen kann. Das erleichert die Datenerhebung, durch das, dass die Daten nur einmal statt zwei Mal aufgeschrieben werden müssen und in ein vorformatiertes Formular eingetragen werden könnten.

Folgende Daten werden jeweils übertragen: Datum des Spiels, Name des Gegners, Anzahl Tore des Gegners, Anzahl Tore HC Keile, Spieler des HC Keile, die das Spiel gespielt haben, Namen der Torschützen aller Tore des HC Keile und der Geber der ersten und zweiten Assists.

- 3. Pflege der Daten

Ein dritter Use case ist die Pflege der Daten. Es soll ermöglicht werden, Einträge hinzuzufügen, zu korrigeiren oder zu löschen. Die Fälle, die hier denkbar sind, sind z.B.: Datum eines Spiels korrigieren, Resultat eines Spiels korrigieren, Namen eines Spielers anpassen, Torschütze/Assistgeber eines Tores korrigieren.

3.3.2. Endpoints

Ein Endpoint ist die URI, also die Adresse, über die eine Ressource aufgerufen wird (Quelle: Traversy Media, YouTube Channel), verbunden mit einer ensprechenden HTTP-Abfrage (z.B. GET, PUT, POST), zwei HTTP-Requests können denselben Endpoint haben, aber verschiedene HTTP-Methoden z.B. GET und POST). Um einen Endpoint zu definieren, muss man die Methdoe (z.B. GET) spezifizieren, sowie den Endpoint,d.h. die URL auf die die Abfrage angewendet wird. Weiter ist zu definieren, welche Daten genau in welchem Format vom Request zurückgegeben werden. Aus diesen Anforderungen lassen sich folgende Endpoints ableiten:

- Ressource: Player
  - GET \\players
  - GET \\players\\{player_id}
  - POST \\players
  - PUT \\players\\{player_id}
  - DELETE \\players\\{player_id}
- Ressource: Game
  - GET \\games
  - GET \\games\\{game_id}
  - POST \\games
  - PUT \\games\\{game_id}
  - DELETE \\games\\{game_id}
- Ressource: Goal
  - GET \\goals

  - POST \\goals
  - PUT \\goals\\{goal_id}
  - DELETE \\goals\\{goal_id}
- Ressource: Opponent
  - GET \\opponents
  - GET \\opponents\\{opponent_id}
  - POST \\opponents
  - PUT \\opponents\\{opopnent_id}
  - DELETE \\opponents\\{opponent_id}

3.4. Gewählte Datenbanktechnologie

Mit dem Spring Framework und Spring Boot 2, respektive der Java Persistence API und ihrer verbreitetsten Implementation Hibernate ist es ganz einfach, eine breite Palette von Datenbank- Technologien zu verwenden. Sie unterstützen die meisten Datenbanken. Man braucht einzig die gewünschten dependcies im der Maven Project Object Model Konfigurationsdatei herunterzuladen und in der Konfigurations-Datei “application-properties”, falls notwendig , die Zugangsdaten (Datenbankname, username, passwort) anzugeben und gegebenenfalls den SQL-Dialekt der Datenbanktechnologie zu spezifizieren, und die JPA und Spring konfigurieren die Verbindung zur Datenbank voll automatisch. Aus reinem Interesse, die open source Datenbank “Postgres” einmal zu verwenden und kennen zu lernen, wurde sie für diese Arbeit gewählt.

4

Struktur und Umsetzung der API

4.1. Technologien und Installation

4.1.1. Installation Spring

Ein Spring-Projekt kann sehr einfach und schnell mit Hilfe des Onlinetools Spring Initializr, , erstellt werden.

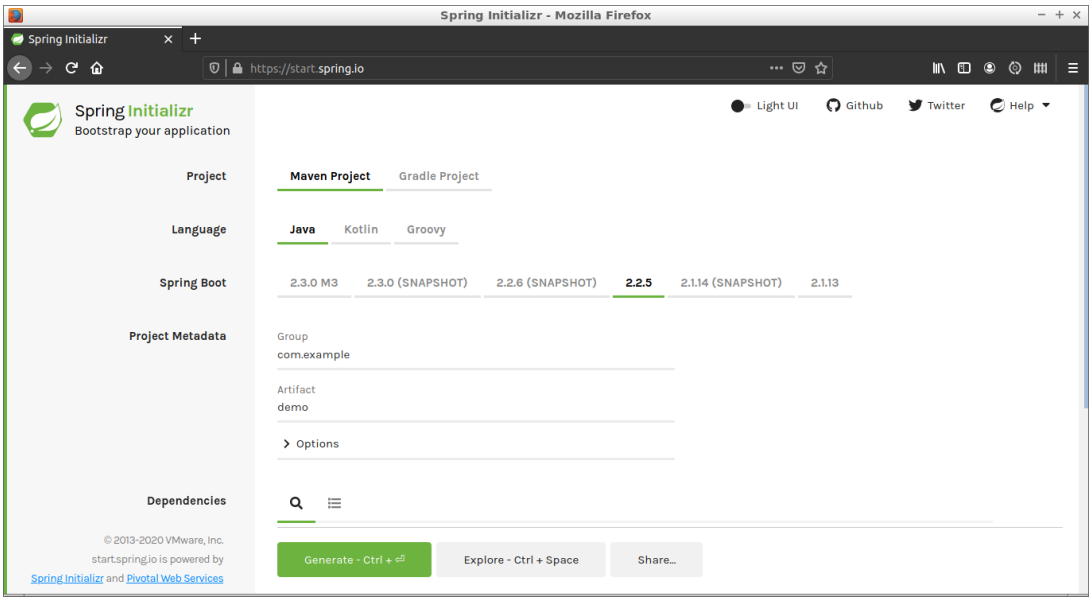

Abbildung 4.1.: Spring Initialzizr Onlinetool

Auf dieser Webseite wählt man zuerst das gewünschte Built-Tool (Maven oder Gradle), die Programmiersprache (Java, Kotlin oder Groovy), die Version von Spring Boot - es wird also bereits davon ausgegangen, dass man Spring Boot nutzt - aus und definiert einen Prpjektnamen und einen Namen für das Paket, zu dem das Projekt gehört. Dann kann man unten bei den Dependencies die nötigen Libraries und Frameworks, die man für sein Projekt braucht, anwählen. Das Resultat lädet man als Jar-File und Maven- respektive Gradleprojekt auf seinen Computer und importiert es in seine IDE. Dort hat man nun eine fertig eingerichtete Projektstruktur zur Verfügung, in welcher bereits die Main-Klasse erstellt ist.

In dem man z.b. das starter web-Paket als Dependency im Spring Initializer wählt, erhält man das Spring MVC Framework, ein Logging Framework, das Spring Core-Framework, sowie ein Validierungsframework in die Applikation integriert. Ähnlich ist es mit Spring Boot starter-JPA. Damit erhält man Java Persistence API (JPA), mit Hibernate eine Standardimplementation von JPA sowie eine automatische Konfiguration dieser Teile der Applikation. [26]

Wenn die Applikation startet, sorgt die Spring Boot Autokonfiguration dafür, dass die Beans im Spring application context (dem Container) erstellt und kofiguriert werden, und somit Spring MVC genutzt werden kann, sowie dass der eingebettete Tomcat-Server konfiguriert und gestartet wird [9, S.26]

Domain Model vs. Entitäten-Beziehungsmodell

Die Literatur zu Hibernate [1, S. 64] beschreibt als Ausgangpunkt 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.[41]

Für diese Arbeit hier würden höchstwahrscheinlich nicht riesige Unterschiede zwischen einem Domain Model und einem Entitäten-Bezihungsmodell bestehen. Bauer und King erwähnen auch, dass Hibernate nicht nur Domain Models aus, sondern auch von Tabellen oder einem anderen Model ausgehend eingesetzt werden kann. [1, S. 64]. Auch wenn bei grösseren Applikationen das Domain Model wohl die bessere Wahl ist, da Hibernate hilft, Objekte in Relationen zu übersetzen und nicht umgekehrt. Daher wird hier als Ausgangspunkt für die Programmierung das Entitäten-Beziehungsmodell wie in Kapitel 3 dargestellt, verwendet.

4.2. Struktur der Applikation

4.2.1. Projektstruktur in Eclipse

Die Ordnerstrukture in der **IDE!** (**IDE!**) ist folgendermassen aufgebaut. Ein Package beinhaltet die Klassen, die Entitäten, die in der Datenbank abgespeichert werden, ein Package beinhaltet die Repository-Interfaces für diese Entitäten und das dritte Package enthält die Controller, welche die Requests an die Enpoints verarbeiten. Auf derselben Ebene wie die Packages ist die Main-Klasse, welche das Programm startet, die Konfiguration liest und die Beans kreiert

4.2.2. Controller

Im MVC-Schema ist die Funktion des Controllers, dass er HTTP-Requests die auf eine spezifische Domain geschickt werden, pro Domain abarbeitet und implementiert, wie die einzelnen Requests verarbetiet werden sollen.

Der Unterschied eines Controllers, der mit der @RestController-Annotation annotiert ist, zu einem traditionellen Spring MVC Controller ist, dass anstatt, dass eine html-Seite gespiesen wird mit den Resultaten der durch den Request aktivierten Methode, nur ein Objekt generiert wird, dessen Daten direkt via http-Response, dem client zurückgegeben werden. [34] Die @RestController Annotation ist eine Abkürzung für @Controller und @ResponseBody und markiert die Klasse als Klasse, bei der jede methode ein Objekt zurückgibt anstatt eine view (html). [34] Sie gibt Spring ebenfalls an, dass der Rückgabewert der Methode direkt in den Response Body der Response gespeist werden soll, anstatt in ein Model für eine View. [9, S. 142]

HTTP-Request-Annotationen

Es ist gute Praxis, auf Klassenlevel bei den Controller-Klassen die RequestMappingAnnotation zu benutzen und dabei den basis URI-Pfad zu definieren, und dann bei den Methoden, welche bei den einzelnen Requests aufgerufen werden, so spezifisch wie möglich

zu sein und zu spezifizieren, um welche Art von Dazu verwendet man die "Get/Post/-Put/DeleteMappingAnnotationen. [9, S. 35]

Die @RestController-Annotation hat zwei Hauptzwecke. Erstens wird mit ihr die Klasse als Component für das component scanning markiert. Zweitens sagt sie Spring, dass die Rückgabewerte aller handler-methoden der Klasse direkt in den Response-Body der HTTP-Response geschrieben werden sollte, anstatt dass sie in das Model gespiesen und eine View daraus generiert werden soll [9]

Wie den Controller für die Post-Methode umsetzen, wenn Informationen, die bereits in der Datenbank sind, z.B. ein Gegner, oder die Listen, auch eingegeben werden müssen? Idee: Klasse bauen, welche die vollständigen Daten inkl. Listen in Datenbank postet, (in repository-Methode). Davor die Kasse füllen mit Rückgaben aus entsprechenden Abfragen aus der Datenbank. Dann Zusammenfügen mit Infos/Klasse, die man im Request-Body, via JSON übergeben hat.

4.2.3. Repositories

Für das Repository-Interface, wie für viele andere Interfaces auch, bietet Spring bereits Standardimplementationen der Methoden, auf dies zurückgreift, wenn der Programmierer keine Implementation spezifiziert. Es reicht also, ein Repository-Interface zu definieren, welches beispielsweise das JpaRepository-Interface erweitert. Dieses hat die Form: JpaRepository<T,ID> [25]. Es muss lediglich der Objekt-Typ der Entität spezifiziert werden, der mit Hilfe dieses Repositories bearbeitet wird sowie der Typ des Identifikationsschlüssels der Entität. Sobald dieses Interface definiert ist, kann man es bereits für die entsprechenden Datenbankmanipulationen verwenden.

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

Listing 4.1: JpaRepository Interface

4.2.4. Entities

Mit der @Entity – Annotation von JPA, die bei einer Klasse angebracht wird, wird angegeben, dass die Klasse einer Tabelle in der Datenbank entspricht. Ihre Felder also den Tabellenspalten. Es braucht für den Primärschlüssel eine @Id-Annotation und die Spalten können auch explizit mit @Column angegeben werden (dabei kann auch der Name der Spalte in der Datenbank präzisiert werden, wenn er von Feldnamen abweicht), müssen aber nicht. Den Primärschlüssel lässt man idealerweise selber vom Framework generieren. Dies kann man tun mit der Annotation @Generated Value

Es braucht zwingend auch einen leeren Konstruktor, auf den Hibernate zurückgreifen kann. Zudem kann man auch einen Konstruktor ohne den Primärschlüssel erstellen, somit kann man einen Eintrag machen, und den Primärschüssel vom Framework erstellen lassen. [22]

Bei einer @OneToOne-Annotation, wird, wenn man die Entität aus der Datenbank lädt ("fetched"), ein join auf den mit als OneToOne und als Fremdschlüssel markierte Tabelle gemacht, und auch alle Werte aus dieser Tabelle gelesen. Dies wird Eager Fetch genannt. [26]

Unideriaktionale Beziehung mit der @OneToOne-Annotation wird eine Beziehung genannt, die in einer Klasse ein Feld mit einer einfach-einfach-Bezieung zu einer anderen Entität als Fremdschlüssel definiert, jedoch in der anderen Klasse nichts vermerkt. Somit kann man von dieser Klasse zwar die Werte, die verbunden sind, aufrufen. Von der anderen Klasse aus kann man aber die zugehörigen Daten aus der anderen Tabelle einzelner Einträge nicht abrufen. Damit das gemacht werden kann, muss die Beziehung bidirektional gemacht werden.

"Owning side of the relationship": Mit der owning side ist diejenige Tabelle gemeint, welche die ID der verbundenen Entität als Fremdschlüssel enthält. Wenn man keine owning side definiert, werden in beiden Tabellen Fremdschlüssel der jeweils anderen Entität abgespeichert. Dies ist Duplikation der Information, was man nicht will in einer Datenbank. Dies kann gelöst werden, indem man in der Tabelle, in der man den Fremdschlüssel nicht eintragen will (auf der "non owning side"), im entsprechenden Feld bei der @OneToOne-Annotation, das Attribut "mappedBy = Name des Felds des Fremdschlüssels in der anderen Tabelle anfügt. [26]

Java Persistence Query Language (JPQL) kann in JPA gebraucht werden, um Queries zu definieren. Es ist ähnlich wie SQL, aber die in den Queries adressierten Entitäten sind nicht die Tabellen in der Datenbank, sondern die Klassen im Java-Code, die mit Entity annotiert sind. Diese Queries werden dann von Hibernate in SQL-Queries umgewandelt, die mit der Datenbank interagieren. [26]

Man kann in Hibernate auch Native Queries, dh. Queries in SQL direkt auf der Datenbank durchführen. Dies kann mit der Entity Manager methode "createNativeQueries" getan werden. Ein Anwendungsfall ist z.B. wenn man viele Tabellen updaten möchte, dies ist mit JPQL nicht oder nur sehr umständlich möglich. Es ist dabei zu beachten, dass der Application Context (die Instanzen der Java Klassen, welche die Datenbank repräsentieren) nicht upgedated werden. Es muss also ein em.refresh() durchgeführt werden, um die Werte aus der Datenbank in den Application Context zu laden. [26]

4.2.5. Main-Klasse

Die Main-Klasse der Applikation heisst „KeileStatsApplication". Sie besteht aus wenig Code. 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". Das ist eine Composite-Annotation, die drei weitere Annotationen umfasst. „@SpringBootConfiguration", die eine spezialisierte Form der @Configuration-Annotation ist und angibt, dass in dieser Klasse javabasierte Konfigurationsinformationen enthält. [9, S. 15]. „@EnablesAutoConfiguration"ermöglicht Spring Boot die Appli-

kation automatisch zu konfigurieren. @ComponentScan ermöglicht component scanning. Das heisst dass andere Klassen mit @Component-Annotationen annotiert werden können und als Komponenten der Applikation im Spring application context registriert werden können. [9, S. 15]

Die Annotation @Component einer Klasse sagt dem Framework, dass diese Klasse eine Bean ist, die instantiiert werden soll. Mit der @SpringBootApplication-Annotation wird Spring mitgeteilt, dass es im Package, in das sich die Klasse mit dieser Annotation befindet, durchsuchen soll danach, welche Beans zu kreieren sind, d.h. nach der @Component-Annotation, somit wird Spring also mitgeteilt wo es nach Beans und ihren Dependencies suchen soll. Der Application Context, der von der Methode run der Klasse SpringBootApplication zurückgegeben wird, managed das Implementieren und erstellen dieser Beans und die Injektion der dependencies. [26]

4.3. 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.

```
<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

Listing 4.2: 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.

```
package ch.keilestats.api.application.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import static springfox.documentation.builders.PathSelectors.regex;

```

```
14 /*Class to configure and enable automatic Swagger2 documentation for the API*/
15 @EnableSwagger2
16 @Configuration
17 public class SwaggerConfig {
18
19     /* Docket on which Swagger2 will run on */
20     @Bean
21     public Docket keileAPI() {
22         return new Docket(DocumentationType.SWAGGER_2).select()
23                 .apis(RequestHandlerSelectors.basePackage("ch.keilestats.api.application
                       ")).paths(regex("/api.*"))
24                 .build().apiInfo(metaInfo());
25     }
26
27     // Titletext to be displayed in the Swagger html-file
28     private ApiInfo metaInfo() {
29
30         ApiInfo apiInfo = new ApiInfo("Keile Stats API",
31                 "API for saving and reading " + "statistics of a just-for-fun Icehockey
                       Team", "1.0",
32                 "Terms of Service", "", "", "");
33
34         return apiInfo;
35     }
36 }
```

Listing 4.3: 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.

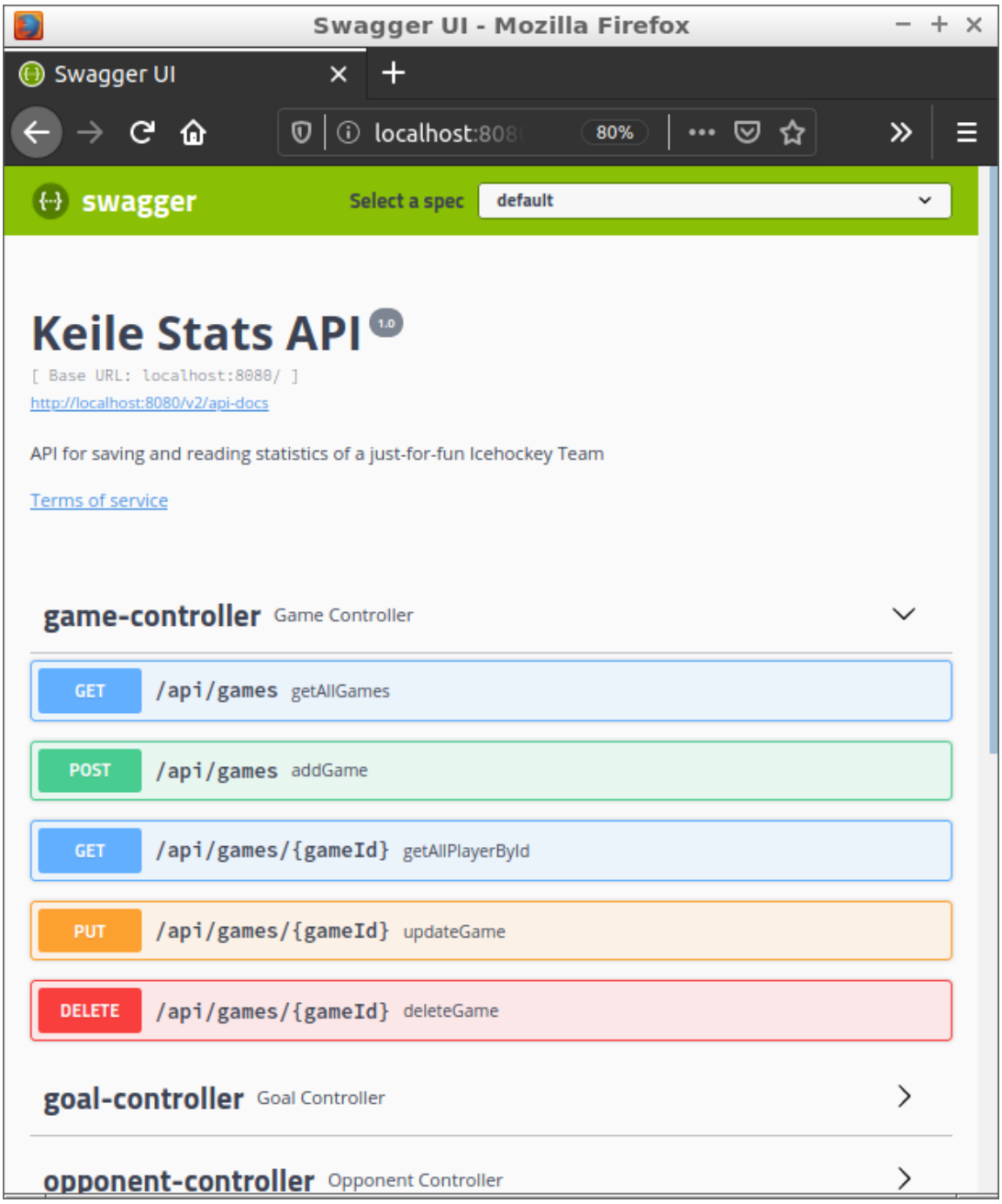

Abbildung 4.2.: Swagger User-Interface

5 Abschliessende Bemerkungen

5.1. Zusammenfassung

5.2. Probleme und Herausforderungen

Schwierigkeiten: post methode implementieren. Verstehen, wie repositories funktionieren.

Verstehen, was genau im Hintergrund geschieht. Dies ist eine sehr grosse Herausforderung in der Arbeit mit dem Spring Framework und Spring Boot als Spring-Neuling und mit nur theoretischen groben Grundkenntnissen der Themen Software-Architektur mit Webservices, Http-Protokoll oder Remote Procedure Calls. Es ist so viel bereits automatisch konfiguriert, wenn man mit dem Spring initaliser und Maven ein Spring Boot Projekt erstellt, dass man zwar sehr schnell eine funktionierende Applikation erstellen kann. Für einen Einsteiger ist jedoch das Programmieren und das Debugging sehr anspruchsvoll, weil, wenn man die Architektur des Spring-Frameworks und Konfiguration und Funktionsweise nicht vertraut ist, man nicht sieht, was jetzt alles genau im Hintergrund abläuft und wie das Programm funktioniert. Es erfordert eine lange Auseinandersetzung mit der Grundstruktur und Grundfunktionsweise des Spring Frameworks und von Spring Boot, um, wenn etwas nicht funktioniert, einen Ansatzpunkt zu haben, wo genau das Problem liegt, auch wenn natürlich die Fehlermeldungen klar sind und helfen. Wenn man mit den Konzepten und Klassen, die in den Fehlermeldungen erwäht sind, nicht vertraut ist, hilft das einem nichts, respektive es braucht ein aufwändiges Recherchieren und vertraut Machen mit dem Spring Framework, bis man einigermassen einen Anhaltspunkt hat, wo der Fehler liegt.

Zudem ist es sehr hilfreich und nötig, sich zuerst theorethische und praktische Grundlagen der Themenbereiche Webservices, Remote Procedure Calls oder HTTP anzueignen, um dann erst überhaupt sich damit zu beschäftigen, wie das Spring-Framework nun diese Herausforderungen angeht. Schwierigkeit auch, datenbankmodell via hibernate respektive Annotationen in java klassen auszudrücken. z.B. many to many relation kann mit annotationen angegeben wer Schaffen, dass die korrekten Daten in die Datenbank eingegeben werden.

A
Abkürzungen

ORM	Objekt/Relationales Mapping
JDBC	Java Database Connectivity
JPQL	Java Persistence Query Language
MVC	Model-View-Controller
EJB	Enterprise Java Beans
ERM	Entity Relationship Model
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
JPA	Java Persistence API
POJO	Plain Old Java Object
POM	Project Object Model
REST	Representational State Transfer
DI	Dependency Injection
AOP	Aspect Oriented Programming
RMI	Remote Method Invocation
API	Application Programming Interface
MVC	Model-View-Controller
ANSI	American National Standards Institute
CSS	Cascading Style Sheet
DBMS	Database Management System
DOM	Document Object Model
HL7	Health Level 7
HTML	Hypertext Markup Language
IoT	Internet of Things
IP	Internet Protocol
IPSec	Internet Protocol Security
JAXB	Java Architecture for XML Binding
JAX-RS	Java API for RESTful Web Services
JPQL	Java Persistence Query Language
JSON	JavaScript Object Notation
JSP	Java Server Pages
JSTL	Java Server Tag Library
NCSA	National Center for Supercomputing Applications
PHP	Hypertext Processor

QR	Quick Response
RFID	Radio Frequency Identification
ROA	Resource Oriented Architecture
SAX	Simple API for XML
SOA	Service Oriented Architecture
SQL	Structured Query Language
SPDY	SPeeDY, an open networking protocol
TCP	Transmission Control Protocol
URI	Unified Resource Identifier
URL	Uniform Resource Locator
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Description Language
WoT	Web of Things
XML	eXtensible Markup Language
XSD	XML Schema Definition

B
Lizenz

Literaturverzeichnis

[1] Christian Bauer and Gavin King. *Hibernate in Action.* Manning, 2005.

[2] Sun Microsystems Inc. JavaBeans. Oracle, 1997.

[3] Rod Johnson. *Expert One-on-One J2EE Design and Development.* Wrox, 2002.

[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.

[5] Jacques Pasquier, Arnaud Durand, and Gremaud Pascal. Lecture notes advanced software engineering, October 2016. Departement für Informatik, Universität Freiburg (CH).

[6] Michael Simons. *Spring Boot 2 - Moderne Softwareentwicklung mit Spring 5.* dpunkt.verlag, 2018.

[7] Stefan Tilkov, Eigenbrodt Martin, Silvia Schreier, and Wolf Oliver. *REST und HTTP - Entwicklung und Integration nach dem Architekturstil des Web.* dpunkt.verlag, 2015.

[8] Craig Walls. *Spring im Einsatz.* Hansen, 2012.

[9] Craig Walls. *Spring in Action.* Manning, 2019.

Referenzen Web

[10] What is the Spring Framework really all about?, Java Brains. https://www.youtube.com/watch?v=gq4S-ovWVlM (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] 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).

[13] Spring Boot Tutorial, How to Do in Java. https://howtodoinjava.com/spring-boot-tutorials (Letzter Aufruf August 27, 2019).

[14] A Comparison Between Spring and Spring Boot. https://www.baeldung.com/spring-vs-spring-boot (Letzter Aufruf August 6, 2019).

[15] Object Messages and Dependencies, YouTube-Kanal Knowledge Dose. https://www.youtube.com/watch?v=W21CLd9zm9k (Letzter Aufruf Sept 17, 2019).

[16] Understanding Dependency Injection, YouTube-Kanal Java Brains. https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot (Letzter Aufruf August 6, 2019).

[17] Difference between Spring and Spring Boot, Dzone. https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot (Letzter Aufruf Oktober 5, 2019).

[18] API Endpoints Tutorial, YouTube Ethan Jarell. https://www.youtube.com/watch?v=C47OXGASGX0 (Letzter Aufruf August 28, 2019).

[19] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (Letzter Aufruf July 30, 2005).

[20] Steps toward the Glory of REST. https://martinfowler.com/articles/richardsonMaturityModel.html (Letzter Aufruf Juni 30, 2019).

[21] Hibernate Framework Basic, Java Beginners Tutorial. https://javabeginnerstutorial.com/hibernate/hibernate-framework-basic/ (Letzter Aufruf Dezember 12, 2019).

[22] JPA and Hinbernate Tutorial for Beginners with Spring Boot and Spring Data JPA, YouTube-Kanal in28minutes. https://www.youtube.com/watch?v=Mal0_XdpdP8 (Letzter Aufruf November 8, 2019).

[23] JSP and Servelts Tutorial : First Java Web Application in 25 steps, YouTube-Kanal in28minutes. https://www.youtube.com/watch?v=Vvnliarkw48 (Letzter Aufruf September 22, 2019).

[24] Spring framework tutorial for beginners with examples in eclipse | Why Spring Inversion of control. https://www.youtube.com/watch?v=r2Q0Jzl2qMQ (Letzter Aufruf November 7, 2019).

[25] Javadoc. https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html (Letzter Aufruf März 13, 2020).

[26] 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).

[27] Pom Reference, Apache Maven Documentation . https://maven.apache.org/pom.html (Letzter Aufruf August 27, 2019).

[28] Spring Rest Docs - Documenting REST API, Example, YouTube-Kanal Java Techie. https://www.youtube.com/watch?v=ghn9p6d___Yc (Letzter Aufruf Februar 3, 2020).

[29] REST principles explained. https://www.servage.net/blog/2013/04/08/rest-principles-explained (Letzter Aufruf Juni 1, 2019).

[30] Building an Application with Spring Boot. https://spring.io/guides/gs/spring-boot/ (Letzter Aufruf August 6, 2019).

[31] How to create a Spring Boot project in Eclipse. https://www.youtube.com/watch?v=WZzGhWSJ6h0 (Letzter Aufruf Juni 30, 2019).

[32] Spring Boot API in 10 Minuten - Tutorial Deutsch. https://www.youtube.com/watch?v=pkVrAmGblXQ (Letzter Aufruf August 6, 2019).

[33] Spring Boot Anwendung mit MySQL Datenbank verbinden - Tutorial Deutsch. https://www.youtube.com/watch?v=TJfwKT-mxOA (Letzter Aufruf August 6, 2019).

[34] Java Spring Online Dokumentation. https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-first-application.html (Letzter Aufruf September 20, 2019).

[35] Spring Tutorial, JavaTPoint. https://www.javatpoint.com/spring-tutorial (Letzter Aufruf Oktober 28, 2019).

[36] REST API Documentation using Swagger2 in Spring Boot, YouTube-Kanal Tech Primers. https://www.youtube.com/watch?v=HHyjWc0ASl8 (Letzter Aufruf Februar 3, 2020).

[37] What is REST API, YouTube-Kanal Telusko. https://www.youtube.com/watch?v=qVTAB8Z2VmA (Letzter Aufruf Juni 30, 2019).

[38] Introduction to Servlets, YouTube-Kanal Telusko. https://www.youtube.com/watch?v=CRvcm7GKrF0 (Letzter Aufruf Sept 19, 2019).

[39] What is Spring Boot? Introduction, YouTube-Kanal Telusko. https://www.youtube.com/watch?v=Ch163VfHtvA (Letzter Aufruf September 23, 2019).

[40] 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).

[41] Domain Model, Wikipeida. https://en.wikipedia.org/wiki/Domain_model (Letzter Aufruf Februar 5, 2020).

[42] JavaBeans, Wikipedia. https://de.wikipedia.org/wiki/JavaBeans (Letzter Aufruf Februar 1, 2020).

[43] Apache Maven, Wikipedia. https://de.wikipedia.org/wiki/Apache_Maven (Letzter Aufruf August 27, 2019).