

LeoTurnier - Turnierverwaltung

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für Informatik

Eingereicht von:

Hain Lukas

Ecker Benjamin

Betreuer:

Franz Gruber-Leitner

Projektpartner:

Thomas Stütz

Leonding, April 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022

L. Hain & B: Ecker

Abstract

A prototype application was developed to simplify the creation and execution of tournaments for the tournament organizer. The tournaments can be run regardless of the sport, and 3 different tournament modes are enabled, KO, Round Robin and Combination. KO means the loser is eliminated from the tournament and the winner goes to the next round. Round Robin means everyone plays against everyone, regardless of results. And Combination means the players are split into a certain number of groups, which is defined by the admin or tournament organizer starting the tournament. In these Groups everyone plays everyone else of that same group, just like in Round Robin, and then a certain number of competitors with the most wins from each group, also defined by the admin or tournament organizer starting the tournament, move up to the KO phase, where the loser is eliminated and the winner plays the next round, just like in KO.

One can access the application either as an admin, tournament organizer or spectator. As an admin you can change, add or delete any kind of data, the tournament organizer is responsible for everything concerning the execution of the tournament, and the spectator cannot change anything in the tournament, but only watch it.

Zusammenfassung

Es wurde ein Prototyp einer Application entwickelt, die das Erstellen und Durchführen von Turnieren für den Turnierorganisator vereinfachen und rationalisieren soll. Die Turniere können unabhängig von der Sportart durchgeführt werden, außerdem werden 3 verschiedene Turniermodi ermöglicht, KO, Round Robin und Combination.

KO bedeutet, dass der Verlierer aus dem Turnier ausscheidet und der Gewinner in die nächste Runde kommt. Round Robin bedeutet, dass jeder gegen jeden spielt, unabhängig vom Ergebnis. Und Combination bedeutet, dass die Spieler in eine bestimmte Anzahl von Gruppen aufgeteilt werden, die der Admin oder der Turnierorganisator zu Beginn des Turniers festlegt. In diesen Gruppen spielt jeder gegen jeden aus der gleichen Gruppe, genau wie bei Round Robin, und dann steigt eine bestimmte Anzahl von Teilnehmern mit den meisten Siegen aus jeder Gruppe, die ebenfalls vom Admin oder Turnierorganisator festgelegt wird, in die KO-Phase auf, in der der Verlierer ausscheidet und der Gewinner in die nächste Runde kommt, genau wie im KO-System.

Man Kann auf die Applikation entweder als Admin, Turnierorganisator oder Zuschauer zugreifen. Als Admin kann man jegliche art von Daten verändern, hinzufügen oder löschen, der Turnierorganisator ist für alles, was die Durchführung der Turniere angeht, verantwortlich, und der Zuschauer kann am Turnier nichts verändern, sondern nur zuschauen.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Gliederung	1
2	Problemstellung	3
2.1	Ausgangssituation	3
2.2	Problemstellung	3
2.3	Aufgabenstellung	3
2.4	Ziele	4
3	Systemarchitektur	5
3.1	Komponenten die Gesamtanwendung	5
3.1.1	Backend	5
3.1.2	Frontend	6
3.2	Verwirklichung der Anforderungen	6
4	Technologien	7
4.1	Java	7
4.2	Quarkus	7
4.2.1	RESTEasy	9
4.2.2	Hibernate ORM with Panache	9
4.2.3	Apache Maven	9
4.3	PostgreSQL	10
4.3.1	Offene Lizenz	11
4.3.2	Großes Potenzial	11
4.3.3	Sichere Replikation	11
4.4	KeyCloak	12
4.4.1	Single Sign-on und Single Sign-out	12
4.4.2	Unterstützung von Standardprotokollen	12
4.4.3	Account Management Console	12

4.4.4	Admin Console	13
4.4.5	Client-Adapter	13
4.5	Docker	13
4.5.1	Vorteile	14
4.5.2	Einschränkungen	15
4.6	Entwicklungsinfrastruktur	15
4.6.1	Git	15
4.6.2	Github	18
4.7	Intellij IDEA	19
4.7.1	Was ist eine IDE?	19
4.7.2	Codeanalyse	20
4.7.3	Versionsverwaltung	20
4.7.4	JVM-Frameworks	20
4.8	WebStorm	21
4.9	Angular	21
4.9.1	Angular Material	21
4.9.2	TypeScript	21
4.9.3	Nginx	22
4.10	MockLab	22
4.10.1	Tutorial	22
5	Umsetzung Frontend	25
5.1	Home-Page	25
5.2	Player-Page	25
5.3	Team-Page	25
5.4	Tournament-Page	25
5.5	Match-Overview-Page	25
6	Umsetzung Backend	26
6.1	Datenmodell	26
6.2	Quarkus Backend	27
6.2.1	Ordnerstruktur	27
6.2.2	Entitätsklassen	29
6.2.3	Repositoryklassen	31
6.2.4	Serviceklassen	33
6.2.5	Turnierdurchführung	35

7 Zusammenfassung	36
Literaturverzeichnis	VII
Abbildungsverzeichnis	IX
Tabellenverzeichnis	X
Quellcodeverzeichnis	XI
Anhang	XII

1 Einleitung

1.1 Gliederung

- Kapitel 1: Einleitung

Hier wird die Diplomarbeit in ihre verschiedenen Kapitel gegliedert, zusammen mit einer kurzen Beschreibung des Kapitels.

- Kapitel 2: Problemstellung

Hier findet man Ausgangssituation, Problemstellung, Aufgabenstellung und Ziele des Projekts beschrieben.

- Kapitel 3: Systemarchitektur

In der Systemarchitektur wird beschrieben, wie die Anforderungen verwirklicht werden und welche Komponenten die Gesamtanwendung hat. Außerdem wird die Aufteilung der Diplomarbeit erläutert.

- Kapitel 4: Technologien

In diesem Kapitel werden alle verwendeten Technologien wie z.B. Programmiersprachen, Frameworks, IDE's aufgelistet sowie erklärt und deren Verwendung begründet.

- Kapitel 5 - Umsetzung Frontend

Die Umsetzung wurde in Frontend und Backend aufgeteilt. Beim Angular Frontend werden Anforderungen, ein kurzes Benutzerhandbuch, verwendete Libs, Vorstellung der Verschiedenen Rollen und ihrer Rechte, etc. erläutert und erklärt.

- Kapitel 6 - Umsetzung Backend

Beim Backend werden Anforderungen, Datenmodell, verwendete Datenbank, Dokumentation der Endpoints, etc. erläutert und gerechtfertigt.

- Kapitel 7: Zusammenfassung

Zuletzt werden die wichtigsten Ergebnisse des Projekts, mögliche Erweiterungen (was könnte aus diesem Projekt noch Thema werden, das hier nicht betrachtet wurde) und eigene Erfahrungen erläutert.

2 Problemstellung

In diesem Kapitel wird beschrieben, um was es in der Diplomarbeit geht und was damit erreicht werden soll.

2.1 Ausgangssituation

Die HTL Leonding ist eine Höhere Technische Lehranstalt. Derzeit wird sie von ca. 1000 Schülern besucht, welche in eine der 4 Zweige:

- Informatik
- Medientechnik
- Elektronik
- Medizintechnik

unterrichtet werden. Die Schulen und Vereine in der Umgebung veranstalten verschiedenste Sportturniere in den Unterschiedlichsten Sportarten und Turniermodi.

2.2 Problemstellung

Derzeit werden an unserer Schule Sportturniere lediglich auf Papier festgehalten, was es relativ umständlich zu verwalten macht und sehr viel Zeit in Anspruch nimmt. Auch sich über den aktuellen Stand eines Turniers zu informieren ist nur mündlich oder an einer Pinnwand möglich.

2.3 Aufgabenstellung

Mit einer neuen Applikation wollen wir dieses bisherige Verfahren ersetzen und die Gestaltung und Verwaltung von Sportturnieren unserer Schule vereinfachen. Dabei soll die Applikation so gestaltet werden, dass sie mehrere Turnier-Modi unterstützt und auch unabhängig von der Sportart ist.

2.4 Ziele

Im Rahmen dieser Diplomarbeit soll nun eine Applikation erstellt werden die das Verwalten und die Informationsbeschaffung eines Turniers komplett digitalisiert und um ein vielfaches beschleunigt und vereinfacht. Die Informationsbeschaffung soll für jede Person mit einem Internetzugang möglich sein, wobei das Verwalten nur ausgewählten Personen vorbehalten ist.

3 Systemarchitektur

3.1 Komponenten die Gesamtanwendung

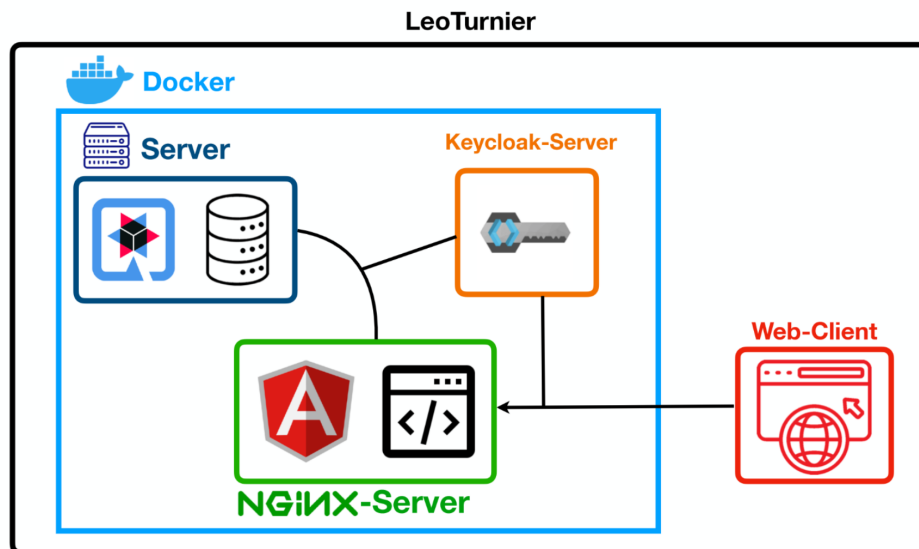


Abbildung 1: System Architecture

Die Diplomarbeit ist Aufgeteilt in Frontend und Backend.

3.1.1 Backend

Im Backend befindet sich eine Quarkus Applikation. Sie wurde in Java geschrieben und verwaltet mithilfe von "Hibernate ORM with Panache" Turnierdaten in einer PostgreSQL Datenbank. Diese Daten werden dem Frontend mithilfe von "RESTEasy" zur Verfügung gestellt. Außerdem befindet sich im Backend ein KeyCloak Server, auf den die Quarkus Applikation mithilfe von "Quarkus OIDC" zugreift. KeyCloak verwaltet Userdaten in einer weiteren PostgreSQL Datenbank. Die Quarkus Applikation, der KeyCloak Server und die beiden PostgreSQL Datenbanken werden in jeweils einem Docker Container ausgeführt.

3.1.2 Frontend

Das Frontend besteht aus einer Angular Applikation. Diese wurde in TypeScript als Code-Behind und HTML als Auszeichnungssprache geschrieben. Für ein schönes und konsistentes User Interface verwendet sie die Library Angular Material. Mithilfe eines KeyCloak Servers wird eine funktionierende Zugriffverwaltung der verschiedenen Benutzer garantiert. Um diesen richtig benützen zu können wird das KeyCloak-Angular Module von Mauricio Vigolo [1] verwendet. Sonstige Funktionen, wie etwa Routing und ein HttpClient, werden fast zur Gänze von der Library Angular Common abdeckt. Die gesamte Angular Anwendung wird dazu noch auf einen Nginx Server deployed welcher in einem Docker Container ausgeführt wird.

3.2 Verwirklichung der Anforderungen

TODO

?

4 Technologien

4.1 Java

Das Backend dieser Applikation wurde in der Programmiersprache Java entwickelt. Java ist nicht nur eine Programmiersprache, sondern ebenso ein Laufzeitsystem, was Oracle durch den Begriff Java Platform verdeutlichen will. So gibt es neben der Programmiersprache Java durchaus andere Sprachen, die eine Java-Laufzeitumgebung ausführen, etwa diverse Skriptsprachen wie Groovy, JRuby, Jython, Kotlin oder Scala. Skriptsprachen auf der Java-Plattform werden immer populärer; sie etablieren eine andere Syntax, nutzen aber die JVM und die Bibliotheken. Zu der Programmiersprache und JVM kommt ein Satz von Standardbibliotheken für Datenstrukturen, Zeichenkettenverarbeitung, Datum/Zeit-Verarbeitung, grafische Oberflächen, Ein-/Ausgabe, Netzwerkoperationen und mehr. Das bildet die Basis für höherwertige Dienste wie Datenbankverbindungen oder Webservices. Integraler Bestandteil der Standardbibliothek seit Java 1.0 sind weiterhin Threads. Sie sind leicht zu erzeugende Ausführungsstränge, die unabhängig voneinander arbeiten können. Mittlerweile unterstützen alle populären Betriebssysteme diese »leichtgewichtigen Prozesse« von Haus aus, sodass die JVM diese parallelen Programmteile nicht nachbilden muss, sondern auf das Betriebssystem verweisen kann. Auf den neuen Multi-Core-Prozessoren sorgt das Betriebssystem für eine optimale Ausnutzung der Rechenleistung, da Threads wirklich nebenläufig arbeiten können. [2]

4.2 Quarkus

Die Programmiersprache alleine reicht allerdings für eine Web-API noch nicht aus, deshalb wird in der Applikation das Quarkus Framework mit den Libraries RESTEasy und Hibernate ORM verwendet. Quarkus ist ein Kubernetes-natives Full-Stack Java Framework für JVMs (Java Virtual Machines) und native Kompilierung, mit dem Java speziell für Container optimiert wird. Es bietet eine effektive Plattform für Serverless-, Cloud- und Kubernetes-Umgebungen. Quarkus wurde speziell für beliebte

Java-Standards, Frameworks und Libraries wie RESTEasy (JAX-RS) und Hibernate ORM (JPA), die wie oben erwähnt in der Applikation verwendet werden, sowie für Apache Kafka, Eclipse MicroProfile, Spring, Spring, Infinispan, Camel und viele mehr konzipiert. Die Dependency-Injection-Lösung von Quarkus basiert auf CDI (Contexts and Dependency Injection) und bietet ein Framework, mit dessen Hilfe die Funktionalität erweitert und ein Framework konfiguriert, gebootet und in Ihre Anwendung integriert werden kann.

Quarkus wurde als benutzerfreundliches Tool konzipiert und integriert Funktionen, die keine oder nur wenig Konfiguration erfordern. Entwickler können die passenden Java Frameworks für ihre Anwendungen auswählen, die dann im JVM-Modus laufen oder kompiliert und im nativen Modus ausgeführt werden können. Das ganz auf Entwickler abgestimmte Quarkus integriert außerdem folgende Funktionen:

- Live-Codierung zur umgehenden Prüfung der Auswirkungen von Code-Änderungen sowie eine schnelle Problembehebung
- Einheitliche imperative und reaktive Programmierung mit einem eingebetteten gemanagten Event Bus
- Einheitliche Konfiguration
- Einfaches Generieren nativer Programmdateien

Ob man die Anwendungen auf einer Public Cloud oder in einem intern gehosteten Kubernetes-Cluster hosten, Eigenschaften wie ein schneller Start sowie ein niedriger Speicherverbrauch sind wichtig, um die Gesamtkosten für das Hosting niedrig zu halten. Quarkus wurde auf Basis einer Container-first-Philosophie entwickelt und soll so einen niedrigen Speicherverbrauch und schnelle Startzeiten auf folgende Weise ermöglichen:

- Erstklassiger Support für Graal/SubstrateVM
- Metadaten-Verarbeitung gleichzeitig mit dem Build
- Reduzierung der Reflektionsnutzung
- Preboot nativer Images

Bei der Anwendungsentwicklung mit Quarkus wird im Vergleich zum traditionellen Java nur ein Zehntel des Speichers belegt. Dazu bietet es bis zu 300 Mal schnellere Startzeiten. Beide Aspekte ermöglichen eine deutliche Reduzierung der Kosten für

Cloud-Ressourcen. Quarkus ist so konzipiert, dass bei der Anwendungsentwicklung der bewährte imperative Code problemlos mit dem nicht-blockierenden reaktiven Code kombiniert werden kann. Dies erweist sich als nützlich sowohl für Java-Entwickler, die an das imperative Modell gewöhnt sind und auch dabei bleiben möchten, und all diejenigen, die einen cloudbativen/reaktiven Ansatz bevorzugen. Das Quarkus-Entwicklungsmodell lässt sich flexibel an alle zu erstellenden Anwendungstypen anpassen. Quarkus ist eine effiziente Lösung zur Ausführung von Java in dieser neuen Welt von Serverless-Architekturen, Microservices, Containern, Kubernetes, Function-as-a-Service (FaaS) und Clouds, weil es für all diese Technologien entwickelt wurde. [3]

4.2.1 RESTEasy

RESTEasy ist ein JBoss/Red Hat-Projekt, das verschiedene Frameworks zur Verfügung stellt, die bei der Erstellung von RESTful Web Services und RESTful Java-Anwendungen unterstützen. Es ist eine Implementierung der Jakarta RESTful Web Services, einer Spezifikation der Eclipse Foundation, die eine Java-API für RESTful Web Services über das HTTP-Protokoll bereitstellt. Darüber hinaus implementiert RESTEasy auch die MicroProfile REST Client Spezifikation API.

RESTEasy kann in jedem Servlet-Container ausgeführt werden, aber eine engere Integration mit WildFly Application Server und Quarkus ist ebenfalls verfügbar, um die Benutzerfreundlichkeit in diesen Umgebungen zu verbessern. [4]

4.2.2 Hibernate ORM with Panache

Bei Hibernate handelt es sich um ein Framework zur Abbildung von Objekten auf relationalen Datenbanken für die Programmiersprache Java - es wird auch als Object Relational Mapping Tool bezeichnet. Hibernate ist in der Programmiersprache Java implementiert und steht als Open Source zur freien Verfügung. Hibernate nutzt das Konzept der Reflexion, um so zur Laufzeit einer Software sogenannte Meta- Informationen über die Struktur von Objekten und die zugrunde liegenden Klassen zu ermitteln, die dann auf die Tabellen einer Datenbank abgebildet werden. [5]

4.2.3 Apache Maven

Als Build-Tool wird im Quarkus Backend Apache Maven verwendet. Maven wurde von der Apache Software Foundation für die Projektverwaltung entwickelt. Entwickler

können damit Java-Projekte automatisieren. Maven ist ein Teil der Apache Software Foundation und wird von dieser gehostet. Das Tool ist aus einem Teil des Jakarta-Projekts hervorgegangen. Maven vereinfacht den Softwareerstellungsprozess, bietet ein einheitliches System für die Entwicklerarbeit, hochwertige Projektinformationen, Richtlinien für das Festlegen von Best Practices und erleichtert die Migration zu neuen Funktionen durch mehr Transparenz. Maven beschreibt sowohl, wie Software gebaut wird, als auch deren Abhängigkeiten. Apache hilft sowohl bei der Verwaltung von Projekten und dient als Tool zum Verständnis. Es hilft also dabei, den Zustand eines Projekts sehr schnell darzustellen. Das Programm wird von Entwicklern und Projektmanagern von Java-Anwendungsprojekten verwendet. Das Tool kann nützlich sein, um den aktuellen Zustand eines Projektes auf einem Blick für technisch weniger versierte Personen, wie Führungspersönlichkeiten und Investoren, zusammenzufassen. Maven basiert auf dem Objektmodell. Projekte werden als eine Pom.xml-Datei gespeichert.

Das Tool verwaltet Projekt-Builds, Reporting und Dokumentation aus der zentralen XML-Informationsbasis. Mit seiner Standard-Plug-in-Architektur kann Maven über Standard-Input mit so gut wie jeder Anwendung zusammenarbeiten. Maven vereinfacht den Prozess für das Erstellen von Java-Anwendungen erheblich, da Nutzer den Status des Projektes viel einfacher einschätzen können. Der Name Maven kommt aus dem Jiddischen und bedeutet Akkumulator von Wissen. Obwohl Maven das Entwickeln von Software anhand von Konventionen ermöglicht, sind reproduzierbare Builds derzeit kein Feature. Die Entwickler arbeiten aber daran. [6]

4.3 PostgreSQL

Die Datenbank der Wahl ist in dieser Diplomarbeit die PostgreSQL Datenbank, da die Diplomanten im Unterricht bereits erfahrung damit angesammelt haben. Dieses DBMS (Datenbank-Managementsystem) bietet eine liberale Lizenz, flexiblen Umgang mit Datentypen, ist skalierbar und vielseitig nutzbar. Der Definition nach ist PostgreSQL ein „objektrelationales“ Datenbank-Managementsystem (DBMS). Es ist Open Source und orientiert sich sehr eng am SQL-Standard. Bestehende SQL-Anwendungen können durch den ANSI-SQL-Standards ohne großen Aufwand integriert werden. Die Funktionsweise entspricht dem klassischen Client-Server-Prinzip, bei dem unterschiedliche Clients Anfragen an den Datenbankserver senden. In der Praxis ist es mit PostgreSQL möglich, komplexere Anwendungen und Workloads zu entwickeln und zu betreiben als mit anderen Lösungen. Es unterstützt eine Vielzahl Datentypen und Operatoren. Entwickler

können hier bei Bedarf eigene Datentypen definieren. PostgreSQL ist nicht nur für Webanwendungen sehr gut geeignet, es kann auch große Enterprise-Datenbanken abbilden. PostgreSQL findet auch in Desktop-Systemen Einsatz: Es ist das Datenbanksystem von Linux und macOS. Diese Systeme liefern das Datenbanksystem standardmäßig mit. [7]

4.3.1 Offene Lizenz

PostgreSQL hat eine Offene Lizenz. Es ist unter der eigenen PostgreSQL-Lizenz veröffentlicht. Generell fallen dabei keine Lizenzkosten an. Diese liberale Lizenz erlaubt es, das System beliebig zu vertreiben. Änderungen müssen nicht der Community zur Verfügung gestellt werden. Die Verwendung von PostgreSQL hat keine nachgelagerten Auswirkungen darauf, wie Weiterentwicklungen und darauf basierende Lösungen verwendet werden dürfen oder veröffentlicht werden müssen. [7]

4.3.2 Großes Potenzial

Die Datenbankgröße ist nicht durch das System begrenzt, sondern nur durch den zur Verfügung stehenden Speicher. Da Postgres flexibel und kompatibel aufgebaut ist, können eine Vielzahl an Anwendungen auf diese Datenbanklösung bauen. Dazu bietet PostgreSQL die Möglichkeit, Trigger einzusetzen und bietet Schnittstellen zu vielen verschiedenen Programmiersprachen, was den Einsatzzweck und die Zusammenarbeit mit unterschiedlichsten Anwendungen potenziell vielfältig macht. [7]

4.3.3 Sichere Replikation

Ein weiterer Punkt ist die detaillierte Zugriffskontrolle und sichere Datenreplikation. In der Praxis ist je nach Anwendungsfall asynchrone oder synchrone Replikation möglich: Synchrone Replikation, um sicherzustellen, dass kritische Transaktionen auf beiden Servern ausgeführt wurden. Asynchrone Replikation für maximale Geschwindigkeit bei weniger kritischen Transaktionen. Ein weiterer Vorteil ist die freie Wahl der Hardware und des Server-OS. PostgreSQL kann auf diversen Linux-Distributionen, BSD, Windows oder macOS gehostet werden. [7]

4.4 KeyCloak

Zum Verwalten von Userdaten wird in dieser Applikation ein KeyCloak Server verwendet. Keycloak ist eine Open-Source-Lösung für Identity und Access Management (IAM), die auf die Entwicklung moderner Anwendungen und Services ausgerichtet ist. [8]

4.4.1 Single Sign-on und Single Sign-out

Möchte man mehrere Applikationen oder Dienste verwenden, so sind in der Regel bei jeder Applikation separate Logins erforderlich. Beim Single Sign-on (SSO) geht es jedoch darum, dass nur eine einzige Authentifizierung notwendig ist (beim ersten Zugriff auf eine der Applikationen). Dies geschieht dadurch, dass die Applikation die Benutzerin bzw. den Benutzer auf eine zentrale Keycloak-Instanz weiterleitet, welche die Login-Maske für die angesprochene Applikation ausgibt und in der Folge die Authentifizierung durchführt. Nach erfolgreicher Anmeldung wird zur Applikation weitergeleitet. Werden anschließend andere Software-Systeme im SSO-Verbund verwendet, dann weiß Keycloak, dass die Benutzerin bzw. der Benutzer bereits authentifiziert ist und fordert keine weitere Anmeldung mehr. Zusätzlich verfügt Keycloak auch über eine Kerberos-Bridge, sodass die Anmeldung am Arbeitsrechner für die Authentifizierung ausreicht. Darüber hinaus unterstützt Keycloak auch ein Single Sign-out: Nach der Abmeldung erfolgt das automatische Ausloggen aus allen Applikationen im SSO-Verbund. [9]

4.4.2 Unterstützung von Standardprotokollen

Keycloak unterstützt Standardprotokolle wie OAuth 2.0 (nur Autorisierung) und das darauf basierende OpenID Connect (Autorisierung und Authentifizierung). Daneben wird mit SAML 2.0 ein weiteres Standardprotokoll unterstützt, welches ähnliche Anforderungen wie OpenID Connect (OIDC) abdeckt. Im Gegensatz zu OIDC existiert SAML jedoch schon länger, basiert auf XML und setzt zum Erreichen von Integrität und Vertraulichkeit digitale Signaturen und entsprechende Zertifikate ein. [9]

4.4.3 Account Management Console

Ein Self-Service-Portal ermöglicht es Benutzern, ihre Daten jederzeit selbst zu verwalten. Keycloak bietet ein solches Portal unter dem Namen „Account Management Console“

an. So können z. B. Passwortänderungen selbständig durchgeführt werden, ohne von Administratoren abhängig zu sein. [9]

4.4.4 Admin Console

Die Admin Console umfasst im Wesentlichen die Konfiguration von:

- Realms (jeweils abgegrenzte Bereiche, für die die anderen in dieser Aufzählung genannten Aspekte einzeln konfiguriert bzw. verwaltet werden können)
- Clients (Applikationen und Dienste)
- Benutzerdatenquellen (siehe User Federation)
- weiteren Identity Providern (siehe Identity Brokering und Social Login)
- die Verwaltung von lokal in Keycloak angelegten Benutzern, Benutzern aus anderen Datenquellen, Benutzergruppen und Rollen

[9]

4.4.5 Client-Adapter

Clients kommunizieren über die angebotenen Standardprotokolle mit Keycloak. Damit diese Kommunikation nicht für jeden Client – evtl. sogar redundant – selbst implementiert werden muss, gibt es für eine Vielzahl an Technologien und Frameworks fertige Client-Adapter, die sich einfach in die eigene Anwendung integrieren lassen und die Kommunikation mit Keycloak übernehmen. [9]

4.5 Docker

Um die Applikation unabhängig von Betriebssystem benutzen zu können, werden das Quarkus Backend, das Frontend, der KeyCloak Server sowie eine PostgreSQL Datenbank für Turnierdaten und eine für Userdaten jeweils in Docker Containern ausgeführt. Die Docker-Technologie verwendet den Linux Kernel und seine Funktionen wie Cgroups und namespaces, um Prozesse zu isolieren, damit diese unabhängig voneinander ausgeführt werden können. Diese Unabhängigkeit ist der Zweck der Container – die Fähigkeit, mehrere Prozesse und Apps getrennt voneinander betreiben zu können. So wird die Infrastruktur besser genutzt und gleichzeitig die Sicherheit bewahrt, die sich aus der

Arbeit mit getrennten Systemen ergibt. Containertools, einschließlich Docker, arbeiten mit einem Image-basierten Bereitstellungsmodell. Das macht es einfacher, eine Anwendung oder ein Paket von Services mit all deren Abhängigkeiten in mehreren Umgebungen gemeinsam zu nutzen. Docker automatisiert außerdem die Bereitstellung der Anwendung (oder Kombinationen von Prozessen, die eine Anwendung darstellen) innerhalb dieser Container-Umgebung. Diese Tools bauen auf Linux-Containern auf und geben den Nutzern so nie dagewesenen Zugriff auf Anwendungen. Sie ermöglichen eine deutlich schnellere Bereitstellung und Kontrolle von Versionen sowie deren Verbreitung. [10]

4.5.1 Vorteile

Modularität

Der Docker-Ansatz für die Containerisierung konzentriert sich darauf, nur einen Teil der Anwendung für eine Reparatur oder Aktualisierung außer Betrieb zu nehmen, ohne dass die gesamte Anwendung außer Betrieb genommen werden muss. Zusätzlich zu diesem auf Microservices basierenden Ansatz können Sie Prozesse in mehreren Apps gemeinsam verwenden – so ähnlich wie bei einer serviceorientierten Architektur (SOA). [10]

Layer und Image-Versionskontrolle

Jedes Docker-Image besteht aus einer Reihe von Layern. Diese Layer werden in einem einzelnen Image vereint. Wenn sich das Image ändert, wird ein Layer erstellt. Jedes Mal, wenn ein Nutzer einen Befehl wie `run` oder `copy` eingibt, wird ein neuer Layer erstellt. Mit Docker werden diese Layer für neuer Container wiederverwendet, was die Entwicklung enorm beschleunigt. Zwischenzeitliche Änderungen werden von den Images gemeinsam verwendet, was die Geschwindigkeit, Größe und Effizienz weiter verbessert. Versionskontrolle ist ein fester Bestandteil des Layering. Bei jeder neuen hat man im Prinzip ein eingebautes Änderungsprotokoll und damit volle Kontrolle über die Container-Images. [10]

Rollback

Das Beste am Layering ist wahrscheinlich das Rollback, also das Zurücksetzen auf die vorherige Version. Jedes Image hat Layers. Wenn man mit der aktuellen Iteration eines

Image nicht zufrieden ist, kann man einfach auf die vorherige Version zurücksetzen. Dieser Ansatz unterstützt eine agile Entwicklung und sorgt, was die Tools angeht, für eine kontinuierliche Integration und Bereitstellung (Continuous Integration/Continuous Deployment - CI/CD). [10]

Schnelle Bereitstellung

Neue Hardware bereitzustellen und zum Laufen zu bringen, dauerte normalerweise Tage und war mit einem enormen Aufwand verbunden. Mit Docker-basierten Containern kann die Bereitstellung auf Sekunden reduziert werden. Indem man für jeden Prozess einen Container erstellt, kann man ähnliche Prozesse schnell mit neuen Apps teilen. Und da zum Hinzufügen oder Verschieben eines Containers das Betriebssystem nicht gebootet werden muss, sind die Bereitstellungszeiten wesentlich kürzer. Darüber hinaus kann man bei dieser Entwicklungsgeschwindigkeit einfach und kosteneffizient Daten erstellen und die von den Containern erzeugten Daten ohne Bedenken löschen. Die Docker-Technologie ist also ein granularer, kontrollierbarer, auf Microservices basierter Ansatz, der deutlich effizienter ist. [10]

4.5.2 Einschränkungen

Docker für sich allein ist für die Verwaltung einzelner Container bestens geeignet. Wenn man beginnt, mehr und mehr Container und containerisierte Apps zu verwenden, die in Hunderte von Bestandteilen zerlegt sind, können die Verwaltung und Orchestrierung sehr schwierig werden. Irgendwann muss man einen Schritt zurückgehen und Container gruppieren, um Dienste wie Vernetzung, Sicherheit, Telemetrie usw. in den Containern bereitzustellen. [10]

4.6 Entwicklungsinfrastruktur

4.6.1 Git

Als Versionskontrollsystem wird von den Diplomanten Git verwendet. Es ist das mit Abstand am weitesten verbreitete Versionskontrollsystem der Welt. Der Name Git wird aus der britischen Umgangssprache übersetzt und bedeutet "Blödmann". [11]

Es ist ein Open-Source Projekt, das ursprünglich 2005 von dem Entwickler des Linux Betriebssystem-Kernels entwickelt wurde. Es ist außerdem sowohl mit Windows als auch

Linux Systemen kompatibel und auch in vielen verschiedenen IDEs integriert. Git ist ein verteiltes Versionskontrollsystem, was bedeutet, dass der Versionsverlauf nicht nur an einem Ort gespeichert ist, wie es bei älteren Versionskontrollsystemen der Fall war, sondern in jeder Arbeitskopie der gesamte Verlauf aller Änderungen im Repository (Aufbewahrungsort) enthalten sind. [12]

Was ist ein Versionskontrollsystem?

Ein Versionskontrollsystem wie Git wird in der Softwareentwicklung verwendet, um Änderungen des Quellcodes zu speichern und zu verwalten. Hier gibt es drei Arten von Versionskontrollsystemen: [12]

- lokale Versionsverwaltung (Local Version Control System - LVCS)

Mit dieser Architektur werden Dateien lokal einfach nur in ein separates Verzeichnis kopiert. [12]

- zentrale Versionsverwaltung (Centralized Version Control System - CVCS)

Hier gibt es einen einen zentralen Server, der alle versionierten Dateien verwaltet und es Personen ermöglicht, Dateien von diesem zentralen Repository abzuholen und auf ihren PC zu übertragen. [12]

- verteilten Versionsverwaltung (Distributed Version Control System - DVCS)

Wie weiter oben schon angesprochen ist diese Variante jene, die von Git benutzt wird. Hier gibt es zwar ein zentrales Repository, aber jede Person hat eine Kopie des Repositories und somit die vollständigen Projektdaten. [12]

Git Befehle

Git verwendet zum verwalten eines Repositories das Terminal, hierzu gibt es einige wichtige Befehle. Zuerst kann man mit "git init" ein leeres Repository erstellen oder ein existierendes nochmal initialisieren, alternativ kann man mit "git clone" ein existierendes Repository kopieren. Damit ein File von Git gespeichert werden kann, muss man es zunächst mit "git add" zum Repository hinzufügen, das ganze kann dann mit "git rm" wieder rückgängig gemacht werden. Mit "git status" wird der momentane Status aller Files im Repository angezeigt, dass heißt, ob das File neu im Repository ist, ob es seit dem letzten Mal Speichern verändert wurde, oder ob es erst garnicht im Repository vorhanden ist. Nun kann man mit "git commit" den momentanen Status des Repositories

als neue Version speichern und mit "git push" an ein "remote" Repository senden. Als letzter wichtiger Befehl gilt noch "git branch", hiermit kann man das Repository in verschiedene "Branches" aufteilen und so neue Features getrennt voneinander entwickeln, und diese dann mit "git merge" im Hauptbranch zusammenfügen. [12]

Leistung

Bei reiner Betrachtung der Leistungsmerkmale von Git, fällt auf, wie stark diese im Vergleich zu vielen Alternativen sind. Für die Commits neuer Änderungen, Branching, Merging und den Vergleich älterer Versionen wurde auf optimale Performance geachtet. Die in Git implementierten Algorithmen schöpfen aus umfassenden Kenntnissen häufiger Attribute echter Quellcode-Dateibäume, deren Modifizierung im Laufe der Zeit und den entsprechenden Zugriffsmustern. [11]

Sicherheit

Bei der Konzipierung von Git gehörte die Integrität des verwalteten Quellcodes zu den obersten Prioritäten. Der Inhalt der Dateien sowie die wahren Beziehungen zwischen den Dateien und Verzeichnissen, Versionen, Tags und Commits – alle diese Objekte im Git-Repository werden mit einem kryptografisch sicheren Hashing-Algorithmus namens SHA1 gesichert. Auf diese Weise sind der Code und der Änderungsverlauf gegen versehentliche als auch vorsätzliche Änderungen geschützt und die vollständige Verfolgbarkeit des Verlaufs wird sichergestellt. [11]

Flexibilität

Eines der wichtigsten Designziele von Git ist die Flexibilität. Git ist in mehrerer Hinsicht flexibel: in seinem Support verschiedener nicht-linearer Entwicklungs-Workflows, in seiner Effizienz sowohl bei kleinen als auch großen Projekten und in seiner Kompatibilität mit vielen bestehenden Systemen und Protokollen. [11]

Warum Git?

Git verfügt über die Funktionen, Performance, Sicherheit und Flexibilität, die den Anforderungen der meisten Teams und einzelnen Entwicklern entsprechen. Auf diese Attribute von Git wurde oben schon eingegangen. In direkter Gegenüberstellung mit den meisten anderen Alternativen schneidet Git sehr gut ab. [11]

4.6.2 Github

Um gemeinsam agil an der Applikation arbeiten zu können, wird von den Diplomanten Github benutzt. Github ist ein Cloud-basiertes Repository Hosting Service, das die verteilte Versionskontrolle von Git zur Verfügung stellt. Es wurde 2008 von Chris Wanstrath, P. J. Hyett, Scott Chacon und Tom Preston-Werner gestartet. Zu dem Zeitpunkt war Git noch relativ unbekannt, weshalb es noch keine anderen Optionen gab. Die Software wurde in der Programmiersprache "Ruby on Rails" und "Erlang" entwickelt. [13]

Das Ziel von Github ist es, eine benutzerfreundliche Oberfläche für Git zur Verfügung zu stellen, mit der man auch mit weniger technischem Wissen die Vorteile von Git ausnutzen kann. Als Unternehmen verdient GitHub Geld, indem es gehostete private Code-Repositories sowie andere geschäftsorientierte Pläne verkauft, die es Unternehmen erleichtern, Teammitglieder und Sicherheit zu verwalten. [14]

Github Actions

GitHub Actions ist ein in Github integriertes Tool, um Prozesse in einem Softwareprojekt zu automatisieren, und wird von den Diplomanten beim Deployment der Applikation benutzt. Dadurch kann man Workflows für dein Repository erstellen. Ein Workflow besteht aus einem oder mehreren Jobs, wobei ein Job aus einem oder mehreren Schritten besteht. Man kann festlegen, ob Workflows in einem Container oder in einer virtuellen Maschine ausgeführt werden sollen. Die Ausführung kann unter den gängigen Betriebssystemen Windows, Linux und macOS erfolgen. Workflows werden durch Events wie beispielsweise ein Push auf das Repository ausgelöst und ausgeführt. Wenn ein Workflow ausgeführt wird, arbeitet er alle seine Jobs, sowie Schritte ab und erstellt dir ein umfangreiches Feedback mit Logs und Ausführungszeiten. Das Feedback kann individuell für jeden Schritt angepasst werden. Eine Action wird in der Web-Oberfläche von GitHub erstellt. Praktischerweise kann eine erstellte Action geteilt und wiederverwendet werden. [15]

Der Workflow in dieser Applikation ist dafür verantwortlich, das Frontend sowie das Backend zu Builden, und anschließend auf die Github Packages Registry zu deployen.

Github Packages

Ein Package ist ein wiederverwendbares Stück Software, das von einer globalen Registry in die lokale Umgebung eines Entwicklers heruntergeladen und in den Anwendungscode integriert werden kann. Da Pakete als wiederverwendbare "Bausteine" fungieren und in der Regel allgemeine Anforderungen erfüllen (z. B. API-Fehlerbehandlung), können sie zur Verkürzung der Entwicklungszeit beitragen. Github Packages ist ein Package Managing Service, der die Veröffentlichung von Packages erleichtert und vollständig in GitHub integriert ist. Alles befindet sich an einem Ort, sodass man zum Suchen und Veröffentlichen von Paketen dieselben Such-, Browsing- und Verwaltungstools verwenden kann wie für Repositories. [16]

4.7 IntelliJ IDEA

IntelliJ IDEA ist eine intelligente, kontextsensitive IDE für Java und andere JVM-Sprachen wie Kotlin, Scala und Groovy, die sich für zahlreiche Anwendungsbereiche eignet, diese Applikation mit eingeschlossen. Auch bei der Entwicklung von Full-Stack-Webanwendungen hilft IntelliJ IDEA Ultimate mit integrierten Tools, Unterstützung für JavaScript und verwandte Technologien sowie erweiterte Unterstützung für gängige Frameworks wie Spring, Spring Boot, Jakarta EE, Micronaut, Quarkus oder Helidon. Darüber hinaus kann IntelliJ IDEA mit Plugins von JetBrains erweitern, um die IDE mit anderen Programmiersprachen wie Go, Python, SQL, Ruby und PHP einzusetzen. [17]

4.7.1 Was ist eine IDE?

Eine IDE (Integrated Development Environment) oder integrierte Entwicklungsumgebung ist Software für die Anwendungsentwicklung, die gängige Entwicklertools in einer zentralen grafischen Oberfläche vereint. Eine typische IDE besteht aus folgenden Komponenten: [18]

Quellcode-Editor

Ein Texteditor, der eine Programmierung von Software-Code mit folgenden Features unterstützt: Syntaxhervorhebung mit visuellen Hinweisen, sprachspezifische Autovervollständigung und eine Bug-Prüfung, während der Code geschrieben wird. [18]

Automatisierung lokaler Builds

Dienstprogramme, mit denen sich einfache wiederholbare Aufgaben im Rahmen der Entwicklung lokaler Software-Builds zur Nutzung durch die Entwickler automatisieren lassen, wie beispielsweise die Kompilierung von Quell- in Binärcode, dessen Paketierung und die Ausführung automatischer Tests. [18]

Debugger

Ein Programm zur Prüfung anderer Programme, mit dem sich die Position von Bugs im Originalcode grafisch anzeigen lässt. [18]

4.7.2 Codeanalyse

IntelliJ bietet intelligente Programmierhilfen an. Durch eine Indizierung des Quellcodes legt die IDE eine virtuelle Karte des Projekts an. Anhand der Informationen in dieser virtuellen Karte kann sie Fehler erkennen, kontextabhängige Completion-Vorschläge anbieten oder Refactorings durchführen. [17]

4.7.3 Versionsverwaltung

IntelliJ IDEA unterstützt die gängigen Versionsverwaltungen (VCS) wie Git, Subversion, Mercurial und Perforce. Man kann ein VCS-Projekt direkt auf dem Startbildschirm klonen, die Unterschiede zwischen zwei Revisionen untersuchen, Branches verwalten, Commits durchführen und pushen, Konflikte bereinigen oder den Verlauf überprüfen. [17]

4.7.4 JVM-Frameworks

IntelliJ IDEA Ultimate bietet Unterstützung für Frameworks und Technologien zur Entwicklung von Anwendungen und Microservices. Die IDE bietet spezielle Unterstützung unter anderem für Quarkus, Spring, Spring Boot, Jakarta EE, JPA und Reactor. [17]

4.8 WebStorm

Obwohl es möglich wäre mit IntelliJ ein Angular Projekt zu entwickeln ist für die Frontend Entwicklung die IDE Webstorm weit angenehmer. Diese ist genauso wie IntelliJ von JetBrains doch enthält mehr support für die Programmiersprachen JavaScript und TypeScript, sowie einen Built-in Debugger für Client-Side JavaScript und Node.js. Besitz man jedoch die Ultimate Version von IntelliJ macht der Gebrauch von Webstorm keinen Sinn, da Ultimate alle oben genannten Features bereitstellt. [19] [20]

4.9 Angular

Angular ist ein Client-Side JavaScript Framework zur Erstellung von Single-Page-Webapplications. Seine komponentenbasierte Architektur welche View und Logik trennt macht es für den Entwickler leicht Applikationen zu warten und testen. Dabei verwendet Angular TypeScript für den Code-Behind und HTML als Auszeichnungssprache. Die vielen gut integrierten Libraries decken Features wie Routing, Verwaltung von Formulare und Client-Server Kommunikation ab.

4.9.1 Angular Material

Angular Material ist eine Library für User Interface Components wie Cards, Inputs und Data Tables welche eine Standard Angular-App sofort aufwerten können. Die Website bietet dazu noch einen guten Überblick der Vielzahl an Möglichkeiten der Components und mithilfe vom Online Live Editor Stackblitz kann diese sofort im Web testen. [21]

4.9.2 TypeScript

Wie der Name schon verrät setzt TypeScript anders als die übliche Variante JavaScript auf strikte Typen und macht somit den Code lesbarer und verständlicher. Wird in JavaScript eine Variable erstellt lässt nur der Name ahnen was im Programm damit passiert, denn es ist nicht notwendig einen Typen zu deklarieren. In TypeScript jedoch muss jede Variable vorher mit einem Typ versehen werden und sollte dies nicht passieren schmeißt der Compiler sofort eine Fehlermeldung. Anders als bei anderen Programmiersprache kann es trotz Fehlern zu einem lauffähigen Programm kommen da der Compiler zwar auf Fehler aufmerksam macht, aber trotzdem eine JavaScript-Datei

erstellt. In den Anfängen von TypeScript gab es noch viele weitere Mehrwerte gegenüber JavaScript, allerdings befinden sich heute beide Programmiersprachen auf dem selben Stand. [22]

4.9.3 Nginx


Nginx ist ein Open-Source-Webserver, der als Webserver, Reverse-Proxy, HTTP-Cache und Load-Balancer verwendet wird. ER zeichnet sich durch seine geringe Speichernutzung und hohe Parallelität aus und verwendet einen asynchronen ereignisgesteuerten Ansatz, bei dem Anforderungen in einem Thread verarbeitet werden. Dabei steuert ein Master-Prozess mehrere Worker-Prozesse, während diese die eigentliche Verarbeitung durchführen und durch die Asynchronität keine anderen Anforderungen blockieren. [23]

4.10 MockLab

MockLab ist ein Service der eine API "mockt" sprich einen Mockup zu deutsch eine Atrappe erstellt, sollte die richtige API noch nicht vorhanden bzw. zurzeit nicht lauffähig sein. Damit ist die Frontend-Entwicklung nicht so stark vom Backend abhängig und eine Fehlerquelle kann durch die immer korrekten Daten des Mockups leichter erfasst werden. [24]

4.10.1 Tutorial

Um mit MockLab zu starten muss zuerst ein Account erstellt werden, oder man loggt sich mit seinem Github Account ein. Einmal eingeloggt werden schon alle erstellten Mock API's angezeigt bzw. beim ersten mal nur die Example Mock API.

Um eine neue Mock API zu erstellen einfach auf den  Button in der Nav-Bar klicken.

Erstellen wir mal eine API die uns eine Fußballmannschaft zurückliefern soll.

Create new mock API

Mock API template



blank
An empty API with no stubs.



mocklab/rest-example
Example mock REST API



mocklab/oauth2-mock
A mock OAuth2 / OpenID Connect login provider.



mocklab/example
Assorted example stubs



mocklab/twitter-oauth1-mock
Basic mock of Twitter's OAuth 1.0a API

API name

Custom hostname (optional)

.mocklab.io

 Save

Die Standard Einstellungen müssten soweit passen, also erstellen wir gleich unseren Stub. Einfach links auf den  **Stubs** klicken und danach in der Menü-Leiste auf  **New**

Nennen wir den Stub einfach mal Fußballmannschaft und definieren einen sinnvollen Pfad für einen GET-Request.

Request

Name

 Add description

GET

▸ Advanced 

▸ Scenario 

Dann müssen wir natürlich noch eine Response für unseren Stub erstellen und antworten einfach mal mit einer JSON-Datei von Paris-Saint-Germain.

Response

Direct **Fault** Proxy

Status

200 ☐ Enable templating ?

+ Header

Content-Type : application/json 

Body

☒ JSON ☐ XML ☐ HTML ☐ Text ☐ Base 64 (binary) <> Indent ✖ Clear

```

1 {
2   "id": 1,
3   "name": "PSG",
4   "totalScore": 9,
5   "players": [
6     {
7       "name": "Sergio Ramos"
8     },
9     {
10      "name": "Neymar"
11    },
12    {
13      "name": "Lionel Messi"
14    }
15  ]
16 }

```

Delay ?

No delay 

Cancel  Save  Clone  Delete

Und siehe schon haben wir eine funktionierende Mockup die Daten beispielsweise für eine Frontend-Entwicklung bereitstellt.

GET  https://3gr41.mocklab.io/football/team Send 

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (6) Test Results 200 OK 490 ms 348 B Save Response 

Pretty Raw Preview Visualize JSON 

```

1 {
2   "id": 1,
3   "name": "PSG",
4   "totalScore": 9,
5   "players": [
6     {
7       "name": "Sergio Ramos"
8     },
9     {
10      "name": "Neymar"
11    },
12    {
13      "name": "Lionel Messi"
14    }
15  ]
16 }

```

Abbildung 2: Postman Request

5 Umsetzung Frontend

5.1 Home-Page

5.2 Player-Page

5.3 Team-Page

5.4 Tournament-Page

5.5 Match-Overview-Page

6 Umsetzung Backend

6.1 Datenmodell

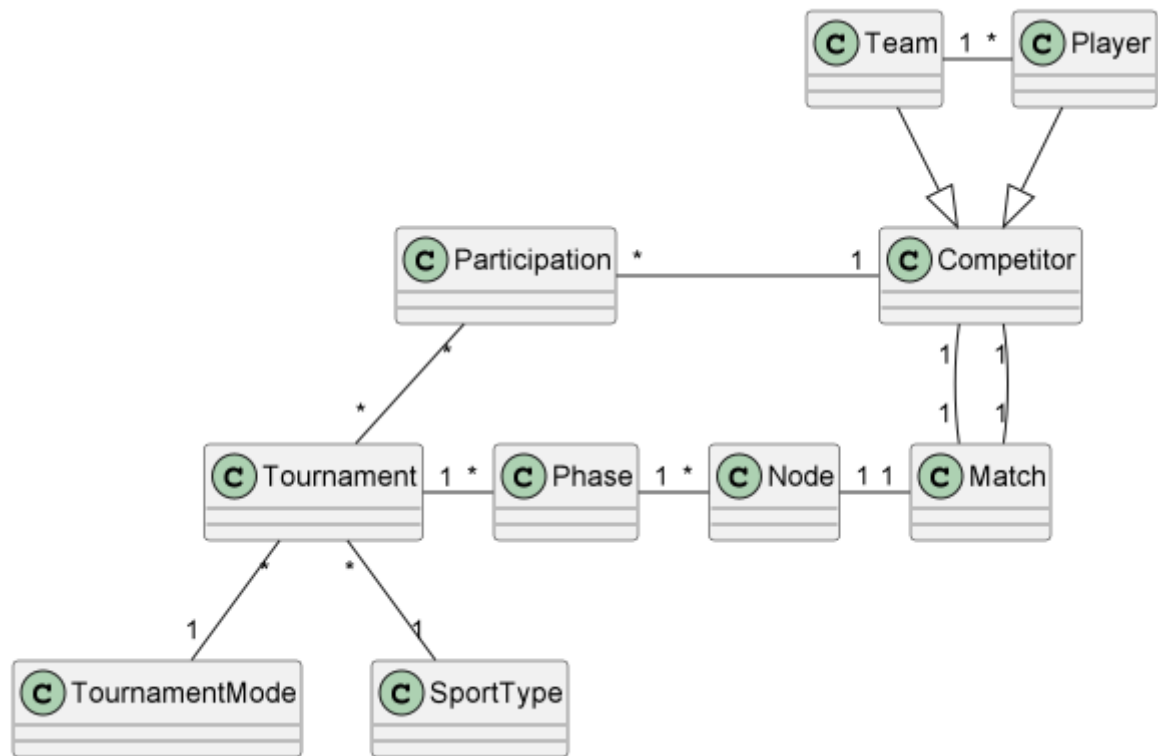


Abbildung 3: Class Diagram

Oben abgebildet ist das Datenmodell des Quarkus Backends. Anfangs erstellt man ein Turnier (Tournament), neben Namen, Startdatum und Enddatum wird hier noch der Turniermodus (TournamentMode) und die Sportart (SportType) festgelegt. Nun gibt man an, welche Teilnehmer (Competitor) an diesem Turnier teilnehmen (Participation). Ein Competitor kann entweder ein einzelner Spieler (Player) oder ein Team von Spielern (Team) sein. Wenn das Turnier gestartet wird, wird das Turnier in Phasen (Phase) unterteilt. In jeder Phase gibt es eine bestimmte Anzahl an Matches (Match), welche jeweils 2 Competitor haben und mithilfe von Nodes (Node) der jeweiligen Phase zugeordnet werden.

6.2 Quarkus Backend

6.2.1 Ordnerstruktur



Abbildung 4: Quarkus Ordnerstruktur

Nach dem erstellen eines Quarkus Projektes ist ein Großteil der Ordnerstruktur bereits im vorhinein gegeben. Das wichtigste File im Projekt befindet sich ganz im äußersten Ordner, nämlich das "pom.xml" File.

```
<?xml version="1.0"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>at.htl</groupId>
  <artifactId>leo-turnier-backend</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <properties>
    <compiler-plugin.version>3.8.1</compiler-plugin.version>
    <maven.compiler.parameters>true</maven.compiler.parameters>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.group-id>io.quarkus.platform</quarkus.platform.group-id>
    <quarkus.platform.version>2.7.1.Final</quarkus.platform.version>
    <surefire-plugin.version>3.0.0-M5</surefire-plugin.version>
  </properties>
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-bom</artifactId>
        <version>2.7.1.Final</version>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
  <build>
    <plugins>
      <plugin>
        <groupId>io.quarkus</groupId>
        <artifactId>quarkus-maven-plugin</artifactId>
        <version>2.7.1.Final</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Abbildung 5: pom.xml

Hier werden nicht nur die Metadaten des Projekts angegeben, sondern auch die Dependencies, also alle externen Libraries, die die Applikation benötigt.

Im "src" Ordner befinden sich ein "main" und ein "test" Ordner. Im "main" Ordner befindet sich der ganze Quellcode der Applikation, und im "test" Ordner befindet sich der Code, der den Quellcode auf seine richtigkeit prüft. Im "main" Ordner befindet sich nun ein "docker" Ordner, auf den später noch eingegangen wird, ein "java" Ordner, in dem sich die Java Klassen der Applikation befindet, und ein "resources" Ordner, in dem weitere Files zu finden sind, auf die der Java Code eventuell zugreifen kann. Außerdem befindet sich im "resources" Ordner das "application.properties" File, in dem verschiedene Quarkus Konfigurationen zu finden sind.

```
# datasource configuration
quarkus.datasource.db-kind = postgresql
quarkus.datasource.username = app
quarkus.datasource.password = app
%prod.quarkus.datasource.jdbc.url = jdbc:postgresql://db:5432/db
%dev.quarkus.datasource.jdbc.url = jdbc:postgresql://localhost:5432/db

# drop and create the database at startup (use `update` to only update the schema)
quarkus.hibernate-orm.database.generation=drop-and-create

# HTTP Config
quarkus.http.cors = true
quarkus.http.root-path=/api

# KeyCloak Config
%prod.quarkus.oidc.auth-server-url=http://auth:8443/realms/leoturnier
%dev.quarkus.oidc.auth-server-url=http://localhost:8443/realms/leoturnier
quarkus.oidc.client-id=leoturnier-client
```

Abbildung 6: application.properties

Die Konfigurationen sind aufgeteilt in Datasource (Datenbank), HTTP (Endpoints) und KeyCloak Konfigurationen.

(Ist der Teil unten zu Basic?) (Der Teil unten gehört noch zur "Quarkus Ordnerstruktur" Abbildung, kann sein dass das etwas unklar ist, wie könnte ich das am besten machen?)

Die Java Klassen im "java" Ordner sind weiters in 4 verschiedene Ordner aufgeteilt. Im "entity" Ordner befinden sich alle Entitätsklassen der Applikation, also jene, die das Datenmodell (siehe Kapitel 6.1 Datenmodell) widerspiegeln. Die Klassen im "repository" sind die Schnittstelle zwischen der Quarkus Applikation und der Datenbank. Sie sind für die CRUD Operationen verantwortlich, also das Speichern, Auslesen, Verändern oder Löschen von Daten nach dem Schema der Entitätsklassen. Außerdem befindet sich hier die Logik hinter den Turnieren. Die Klassen im "boundary" sind die Schnittstelle zum Frontend. Sie nehmen Requests, also Anforderungen, von außen an, führen die

jeweilige Methode aus den Repository Klassen aus und liefern dann eine Response, also eine Antwort, zurück. Zuletzt gibt es noch die Klassen in "dto" Ordner, diese sind Klassen, in denen Daten gespeichert werden können, die für die Logik in den Repository Klassen benötigt wird, jedoch nicht unbedingt Teil des Datenmodells sind.

6.2.2 Entitätsklassen

```
@Entity
@Table(name = "T_TOURNAMENT")
public class Tournament {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "T_SEQ")
    @Column(name = "T_ID")
    Long id;

    @Column(name = "T_NAME")
    String name;

    @Column(name = "T_START_DATE")
    LocalDate startDate;

    @Column(name = "T_END_DATE")
    LocalDate endDate;

    @ManyToOne
    @JoinColumn(name = "T_ST_ID")
    SportType sportType;

    @ManyToOne
    @JoinColumn(name = "T_TM_ID")
    TournamentMode tournamentMode;

    @Column(name = "T_IS_FINISHED")
    boolean isFinished;
```

Abbildung 7: Entitätsklasse

Oben abgebildet ist eine der Entitätsklassen dieser Applikation, nämlich die Tournament Klasse. Ganz oben, über der Klasse, ist die Annotation "@Entity" zu finden, um sicherzustellen, dass die Klasse von JPA als Entitätsklasse erkannt wird, und das Library dafür eine Tabelle erstellt. Darunter befindet sich die "@Table" Annotation, die der Tabelle in der Datenbank ihren Namen gibt. Das Schema der Namensgebung von Tabellen lautet wie folgt: am Anfang steht immer die Abkürzung für den Tabellennamen, in diesem Fall ist das "T", danach kommt ein Unterstrich, gefolgt von dem Namen der Entitätsklasse in Großbuchstaben. In der Klasse selbst befinden sich nun die

Klassenvariablen, welche von der Annotation "@Column" ihren Namen in der Datenbank verleiht. Das Schema der Namensgebung ist ähnlich, wie das der Tabellen: Anfangs die Abkürzung des Tabellennamens, dann ein Unterstrich als Trennung, gefolgt vom Namen der Klassenvariable, ebenfalls in Großbuchstaben. Die ID hat eine extra annotation, nämlich "@Id", das bedeutet dass sie der Primärschlüssel der Entitätsklasse ist. Außerdem steht bei den Variablen "tournamentmode" und "sportType" noch die "@ManyToOne" Annotation. Diese Felder werden dann in der Datenbank zu den Fremdschlüsseln auf andere Tabellen. Hier lautet das Namensschema wie folgt: Zuerst die Abkürzung der eigenen Klasse, dann der Name des Primärschlüssels aus der Tabelle, auf die der Fremdschlüssel zeigt, wieder, getrennt durch einen Unterstrich. Es gibt 4 Verschiedene Annotationen, die eine Beziehung zu anderen Tabellen herstellen:

- @OneToOne
- @ManyToOne
- @OneToMany
- @ManyToMany

(wie könnte ich unten noch zu der "anderen" Klasse sagen?)

Die erste Mengenangabe steht immer für die eigene Klasse, die zweite steht für die andere Klasse. Diese braucht in diesem Fall natürlich auch die "@Entity" Annotation sowie einen Primärschlüssel. Bei "@OneToOne" und "@ManyToOne" wird in der Datenbank der Fremdschlüssel in der eigenen Tabelle erstellt, bei "@OneToMany" wird der Fremdschlüssel in der anderen Tabelle erstellt (hier muss die Variable, bei der die Annotation steht, auch eine Collection sein), und bei "@ManyToMany" wird eine Assotiationstabelle erstellt, in der sich dann 2 Fremdschlüssel befinden: einmal der auf die Tabelle der eigenen Klasse, und einmal die der anderen Klasse.

Unter den Klassen befinden sich dann standardmäßig noch der Constructor sowie die Getter und Setter.

6.2.3 Repositoryklassen

```

@ApplicationScoped
@Transactional
public class TournamentRepository implements PanacheRepository<Tournament> {

    @Inject
    ParticipationRepository participationRepository;

    @Inject
    TournamentModeRepository tournamentModeRepository;

    @Inject
    SportTypeRepository sportTypeRepository;

    @Inject
    PhaseRepository phaseRepository;

    public Tournament add(Tournament tournament) {...}

    public Tournament modify(long id, Tournament tournament) {...}

    public Tournament getById(Long id) { return find( query: "id", id).firstResult(); }

    public List<Tournament> getByCompetitorId(Long competitorId) {...}

    public List<Tournament> getAll() { return listAll(); }

    public Tournament delete(Long id) {...}

    public long clear() {...}
}

```

Abbildung 8: Repositoryklasse

Wie oben schon erwähnt sind die Repositoryklassen die Schnittstelle zwischen Quarkus Backend und Datenbank, dies wird durch die "Hibernate ORM with Panache" Library ermöglicht. Außer ein paar Ausnahmen befinden sich hier nur die CRUD (Create - Read - Update - Delete) Operationen, als Create Operation fungiert hier die "add" Methode, als Read Operationen die "get" Methoden (getById, getByCompetitorId, getAll), als Update Operation die "modify" Methode und als Delete Operationen die "delete" und "clear". Über der Klasse befinden sich die Annotation "@ApplicationScoped" und "@Transactional". "@ApplicationScoped" ermöglicht es anderen Repositoryklassen, mithilfe von "@Inject" eine Instanz dieser Klasse zu injizieren, genau so, wie es diese Klasse weiter unten auch tut. Es wird zum Beispiel in der "add" Methode "TournamentRepository" Klasse (siehe Abbildung "add Methode aus TournamentRepository" unten) die der "SportTypeRepository" Klasse benötigt, da sich in der "Tournament"

Klasse ein Fremdschlüssel auf die "SportType" Klasse befindet. Sollte also die Sportart, auf die dieser Fremdschlüssel zeigt, in der Datenbank nicht existieren, kommt es zu einer SQLException, weshalb in der add Methode von "TournamentRepository", bevor das Turnier selbst mit "persist" gespeichert wird, noch die Sportart mithilfe der add Methode der "SportTypeRepository" Klasse in der Datenbank.

```
public Tournament add(Tournament tournament) {
    if (tournament == null) {
        return null;
    }
    Tournament existing = getById(tournament.getId());
    if (existing != null) {
        return existing;
    }
    if (tournament.getTournamentMode() != null) {
        tournament.setTournamentMode(
            tournamentModeRepository.getById(tournament.getTournamentMode().getId()));
    }
    tournament.setSportType(
        sportTypeRepository.add(tournament.getSportType()));

    persist(tournament);
    return tournament;
}
```

Abbildung 9: add Methode aus TournamentRepository

Das gleiche passiert bei der "modify" Methode, wenn der Fremdschlüssel auf eine Sportart zeigt, die noch nicht existiert.

"@Transactional" wird verwendet, um in jeder Methode dieser Klasse vor den Ausführen eine neue Transaktion zu erstellen, und diese danach zu commiten. Wenn Daten in der Datenbank gespeichert, modifiziert oder gelöscht werden, werden diese Änderungen erst dann wirklich in die Datenbank übernommen, wenn eine Transaktion commitet wurde.

Das PanacheRepository ist eine Erweiterung aus dem "Hibernate ORM **with Panache**" Library, ohne Panache wäre theoretisch auch alles, was in den Repositoryklassen Implementiert wurde, möglich, jedoch etwas umständlicher. Zum Beispiel gibt es ohne Panache die "persist" Methode nicht, in dem Fall müsste man eine "EntityManager" Instanz mithilfe von Dependency Injection injizieren, in davon dann die "persist" Methode hernehmen. Auch andere Methoden, wie zum Beispiel "find" oder "listAll" wären ohne Panach nicht verfügbar, und müssten vom Entwickler selbst Implementiert werden.

6.2.4 Serviceklassen

```

@Path("/tournament")
public class TournamentService {

    @Inject
    TournamentRepository repository;

    @POST
    @RolesAllowed({"admin"})
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response add(Tournament tournament, @Context UriInfo info) {...}

    @PUT
    @RolesAllowed({"admin"})
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes(MediaType.APPLICATION_JSON)
    public Response modify(@QueryParam("id") Long id, Tournament tournament, @Context UriInfo info) {...}

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Response get(@QueryParam("id") Long id, @QueryParam("competitorId") Long competitorId) {...}

    @DELETE
    @RolesAllowed({"admin"})
    @Produces(MediaType.APPLICATION_JSON)
    public Response delete(@QueryParam("id") Long id) { return Response.ok(repository.delete(id)).build(); }
}

```

Abbildung 10: Serviceklasse

Die Serviceklassen sind die Schnittstelle zum Frontend, diesem stellen sie durch Endpoints, welche von Requests angesprochen werden, Daten aus der Datenbank zur Verfügung, dies wird durch die "RESTEasy" Library ermöglicht. Über der Klasse wird mithilfe der Annotation "@Path" der URL Path zu den Endpoints dieser Klasse festgestellt. In der Klasse selbst wird zuerst eine Instanz des dazugehörigen Repositories injiziert. Danach folgt für jede CRUD Operation ein Endpoint. Es gibt verschiedene Arten von Endpoints, in dieser Applikation werden hauptsächlich 4 verwendet:

- "@POST" bei Create Operationen
- "@PUT" bei Update Operationen
- "@GET" bei Read Operationen
- "@DELETE" bei Delete Operationen

"@RolesAllowed" gibt an, welche Rollen zugriff auf diesen Enpoint haben. Darauf wird im Kapitel KeyCloak noch genauer eingegangen. "@Consumes" und "@Produces" geben an, welche Art von Content in den Requests (@Consumes) bzw. den Responses (@Produces) übergeben werden soll. Außerdem wäre es noch möglich, den URL Path zu einem bestimmten Endpoint zu ändern, und zwar wieder mit der "@Path" Annotation. Bei den Parametern der Methoden gibt es manche mit der Annotation "@QueryParam", diese

Annotation gibt an, dass dieser Parameter von der Request in der URL definiert wird. Zum Beispiel sieht die URL einer Request auf den GET Endpoint wie folgt aus:

http://localhost:8080/api/tournament?id=1

Der Hostname ist in diesem Quarkus Backend "localhost", der Port 8080, der Root Path "api" wurde im "application.properties" File festgelegt, und der Path "tournament" wie erwähnt mit der "@Path" Annotation. Nach dem "?" werden immer die Queryparameter ("@QueryParam") übergeben, in diesem Fall die "id" mit dem Wert 1. Der Parameter "UriInfo" wird benötigt, um im Header der Response den Path zurückzugeben, mit dem man den neu hinzugefügten bzw. modifizierten Datensatz finden kann.

Was bei den Serviceklassen noch auffallend ist, ist dass es zum Beispiel für Read Operationen insgesamt nur einen Endpoint pro Serviceklasse gibt, obwohl es in den Repositoryklassen mehrere Get Methoden gibt (siehe Abbildung "Repositoryklasse"). Dies wird dadurch gelöst, dass Endpoints auch erfolgreich ausgeführt werden, wenn nicht alle Queryparameter in der URL definiert wurden, diese haben standardmäßig den Wert "null". Mit diesem Prinzip wird erreicht, dass mit dem gleichen URL Path verschiedene Read Operationen durchführen kann.

```
@GET
@Produces(MediaType.APPLICATION_JSON)
public Response get(@QueryParam("id") Long id, @QueryParam("competitorId") Long competitorId) {
    if (id != null) {
        return Response.ok(repository.getById(id)).build();
    } else if (competitorId != null) {
        return Response.ok(repository.getByCompetitorId(competitorId)).build();
    }
    return Response.ok(repository.getAll()).build();
}
```

Abbildung 11: get Methode aus TournamentService

Im obigen Beispiel hat man 3 Möglichkeiten:

- keinen Query Parameter anzugeben, um alle Turniere auszulesen
- den "id" Query Parameter anzugeben, um nur das Turnier mit der angegebenen Id auszulesen
- den "competitorId" Query Parameter anzugeben, um alle Turniere auszulesen, in denen der Competitor mit der angegebenen Id dabei war.

Die einzige Entitätsklasse, für die es nur READ Endpoints gibt, ist die TournamentMode Klasse, Turniermodi werden beim Start des Backends mithilfe der "InitBean" Klasse

automatisch in die Datenbank hinzugefügt, da diese für die Turnierdurchführung jeweils einzeln implementiert werden müssen.

6.2.5 Turnierdurchführung

Für die Turnierdurchführung gibt es insgesamt 4 verschiedene Repositoryklassen, eine für jeden Turniermodus (Elimination, Round Robin, Combination), und eine, die dafür zuständig ist, die Methoden aus dem richtig

7 Zusammenfassung

Aufzählungen:

- Itemize Level 1
 - Itemize Level 2
 - Itemize Level 3 (vermeiden)
- 1. Enumerate Level 1
 - a. Enumerate Level 2
 - i. Enumerate Level 3 (vermeiden)

Desc Level 1

Desc Level 2 (vermeiden)

Desc Level 3 (vermeiden)

Literaturverzeichnis

- [1] M. Vigolo, „Keycloak-Angular,” 2022, letzter Zugriff am 24.08.2022. Online verfügbar: <https://www.npmjs.com/package/keycloak-angular>
- [2] C. Ullenboom, *Java ist auch eine Insel*, 15. Aufl. Galileo Computing, 2020.
- [3] , „Was ist Quarkus?” 2020, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.redhat.com/de/topics/cloud-native-apps/what-is-quarkus>
- [4] —, „RESTEasy,” 2022, letzter Zugriff am 20.08.2022. Online verfügbar: <https://resteasy.dev/>
- [5] Redaktion ComputerWeekly.de, „Hibernate,” 2020, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.itwissen.info/Hibernate-hibernate.html>
- [6] , „Maven,” letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.computerweekly.com/de/definition/Maven>
- [7] Patrick Woods, „Was ist PostgreSQL?” 2021, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.plusserver.com/blog/was-ist-postgresql>
- [8] , „Keycloak – Open Source Identity und Access Management,” 2018, letzter Zugriff am 21.08.2022. Online verfügbar: <https://login-master.com/keycloak-open-source-identity-und-access-management/>
- [9] Tobias Surmann, „Keycloak – Ein Überblick,” letzter Zugriff am 21.08.2022. Online verfügbar: <https://www.smf.de/keycloak-ein-ueberblick/>
- [10] , „Docker – Funktionsweise, Vorteile, Einschränkungen,” 2018, letzter Zugriff am 20.08.2022. Online verfügbar: https://www.redhat.com/de/topics/containers/what-is-docker?sc_cid=7013a000002wLw0AAE&gclid=Cj0KCQjwjIKYBhC6ARIsAGEds-KwmuTopp7YzOB18IVrXiFvqwtPWlxMCQlrUTA9mDJokqoQZRqMcNcaAv5LEALw_wcB&gclsrc=aw.ds
- [11] —, „Was ist Git?” letzter Zugriff am 18.08.2022. Online verfügbar: <https://www.atlassian.com/de/git/tutorials/what-is-git>
- [12] —, „GIT FACHBEGRIFFE EINFACH ERKLÄRT,” letzter Zugriff am 18.08.2022. Online verfügbar: <https://www.arocom.de/fachbegriffe/webentwicklung/git>
- [13] Saurabh Mhatre, „The untold story of Github,” 2016, letzter Zugriff am 18.08.2022. Online verfügbar: <https://smhatre59.medium.com/the-untold-story-of-github-132840f72f56>
- [14] , „Was ist GitHub? Eine Einführung in GitHub für Einsteiger,” 2020, letzter Zugriff am 18.08.2022. Online verfügbar: <https://kinsta.com/de/wissensdatenbank/was-ist-github/>

- [15] Cem Caylak, „GitHub Actions im Java Projekt,” 2020, letzter Zugriff am 19.08.2022. Online verfügbar: <https://www.adesso.de/de/news/blog/github-actions-im-java-projekt.jsp#:~:text=GitHub%20Actions%20ist%20ein%20hauseigenes,einem%20oder%20mehreren%20Schritten%20besteht>.
- [16] Liz Parody, „GitHub Package Registry: Pros and Cons for the Node.js Ecosystem,” 2019, letzter Zugriff am 19.08.2022. Online verfügbar: <https://nodesource.com/blog/github-package-registry/>
- [17] , „Was ist IntelliJ IDEA?” letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.jetbrains.com/de-de/idea/features/>
- [18] —, „Was ist eine IDE?” 2019, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.redhat.com/de/topics/middleware/what-is-ide>
- [19] —, „Any benefit of using Webstorm vs IntelliJ Ultimate?” 2020, letzter Zugriff am 23.08.2022. Online verfügbar: <https://intellij-support.jetbrains.com/hc/en-us/community/posts/360007008299-Any-benefit-of-using-Webstorm-vs-IntelliJ-Ultimate->
- [20] K. Gupta, „IntelliJ IDEA vs PhpStorm vs WebStorm IDE Differences,” 2018, letzter Zugriff am 23.08.2022. Online verfügbar: <https://www.freelancinggig.com/blog/2018/05/15/intellij-idea-vs-phpstorm-vs-webstorm-ide-differences/>
- [21] O. Mensah, „Creating Beautiful Apps with Angular Material,” 2019, letzter Zugriff am 22.08.2022. Online verfügbar: <https://auth0.com/blog/creating-beautiful-apps-with-angular-material/%7D>
- [22] J. Tillmann, „Was ist eigentlich Typescript,” 2017, letzter Zugriff am 22.08.2022. Online verfügbar: <https://t3n.de/news/eigentlich-typescript-859869/>
- [23] , „Was ist Nginx? Ein grundlegender Blick darauf, was es ist und wie es funktioniert,” 2022, letzter Zugriff am 23.08.2022. Online verfügbar: <https://kinsta.com/de/wissensdatenbank/was-ist-nginx/>
- [24] —, „Was ist Mocklab,” letzter Zugriff am 22.08.2022. Online verfügbar: <https://get.mocklab.io/%7D>

Abbildungsverzeichnis

1	System Architecture	5
2	Postman Request	24
3	Class Diagram	26
4	Quarkus Ordnerstruktur	27
5	pom.xml	27
6	application.properties	28
7	Entitätsklasse	29
8	Repositoryklasse	31
9	add Methode aus TournamentRepository	32
10	Serviceklasse	33
11	get Methode aus TournamentService	34

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang