

# 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

wir erklären an Eides statt, dass wir 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 haben.

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

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Problemstellung</b>	<b>3</b>
2.1	Ausgangssituation . . . . .	3
2.2	Problemstellung . . . . .	3
2.3	Aufgabenstellung . . . . .	3
2.4	Ziele . . . . .	4
<b>3</b>	<b>Systemarchitektur</b>	<b>5</b>
3.1	Komponenten die Gesamtanwendung . . . . .	5
3.1.1	Backend . . . . .	5
3.1.2	Frontend . . . . .	6
3.1.3	KeyCloak . . . . .	6
<b>4</b>	<b>Technologien</b>	<b>7</b>
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 . . . . .	10
4.3	PostgreSQL . . . . .	11
4.3.1	Offene Lizenz . . . . .	11
4.3.2	Großes Potenzial . . . . .	11
4.3.3	Sichere Replikation . . . . .	12
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 . . . . .	13
4.4.4	Admin Console . . . . .	13

4.4.5	Client-Adapter . . . . .	13
4.5	Docker . . . . .	14
4.5.1	Vorteile . . . . .	14
4.5.2	Einschränkungen . . . . .	15
4.6	Angular . . . . .	16
4.6.1	Angular Material . . . . .	16
4.6.2	TypeScript . . . . .	16
4.6.3	Nginx . . . . .	16
4.7	Entwicklungsinfrastruktur . . . . .	17
4.7.1	Git . . . . .	17
4.7.2	Github . . . . .	19
4.7.3	Intellij IDEA . . . . .	21
4.7.4	WebStorm . . . . .	22
4.8	Postman . . . . .	22
4.9	MockLab . . . . .	23
4.9.1	Tutorial . . . . .	23
<b>5</b>	<b>Benutzerhandbuch</b>	<b>27</b>
5.1	Home-Page - Rollen:Alle . . . . .	27
5.2	Tournament-View-Page - Rollen: Alle . . . . .	30
5.2.1	Tree-View . . . . .	30
5.2.2	Table-View . . . . .	31
5.3	Login-Page - Rollen: Admin,Tournament-Organizer . . . . .	32
5.4	Player-Page - Rollen:Admin,Tournament-Organizer . . . . .	33
5.4.1	Player-Details-Page . . . . .	33
5.5	Team-Page - Rollen:Admin,Tournament-Organizer . . . . .	34
5.5.1	Team-Details-Page . . . . .	35
5.6	Tournament-Page Rollen:Admin,Tournament-Organizer . . . . .	36
5.6.1	Tournament-Details-Page . . . . .	37
5.7	Match-Overview-Page - Rollen:Admin,Tournament-Organizer . . . . .	39
5.7.1	Match-Submission-Page . . . . .	40
<b>6</b>	<b>Umsetzung Frontend</b>	<b>42</b>
6.1	Ordnerstruktur . . . . .	42
6.1.1	App Components . . . . .	44
6.1.2	Hilfsklassen . . . . .	46

6.1.3	Components . . . . .	48
6.2	Tournament-Tree . . . . .	50
<b>7</b>	<b>Umsetzung Backend</b>	<b>52</b>
7.1	Datenmodell . . . . .	52
7.2	Quarkus Backend . . . . .	53
7.2.1	Ordnerstruktur . . . . .	53
7.2.2	Entitätsklassen . . . . .	56
7.2.3	Repositoryklassen . . . . .	58
7.2.4	Serviceklassen . . . . .	60
7.2.5	Turnierdurchführung . . . . .	62
7.2.6	Elimination . . . . .	62
7.2.7	Round Robin . . . . .	71
7.2.8	Combination . . . . .	75
7.3	KeyCloak . . . . .	77
<b>8</b>	<b>Deployment</b>	<b>79</b>
8.1	Backend . . . . .	79
8.2	Frontend . . . . .	80
8.3	Workflow . . . . .	81
8.4	Docker-Compose . . . . .	82
<b>9</b>	<b>Zusammenfassung</b>	<b>85</b>
	<b>Literaturverzeichnis</b>	<b>VII</b>
	<b>Abbildungsverzeichnis</b>	<b>X</b>
	<b>Tabellenverzeichnis</b>	<b>XII</b>
	<b>Quellcodeverzeichnis</b>	<b>XIII</b>
	<b>Anhang</b>	<b>XIV</b>

# 1 Einleitung

Diese Arbeit ist in folgende Kapitel gegliedert.

- 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: Benutzerhandbuch

jaja bruder

- Kapitel 6: 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 7: Umsetzung Backend

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



- Kapitel 8: Deployment

In diesem Kapitel wird erklärt, wie genau die fertig Applikation einsatzbereit gestellt wurde.

- Kapitel 9: 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.

Nun da die Ziele der Diplomarbeit definiert sind, geht es weiter mit der Systemarchitektur.

# 3 Systemarchitektur

Dieses Kapitel befasst sich mit dem Aufbau und den Komponenten der Applikation.

## 3.1 Komponenten die Gesamtanwendung

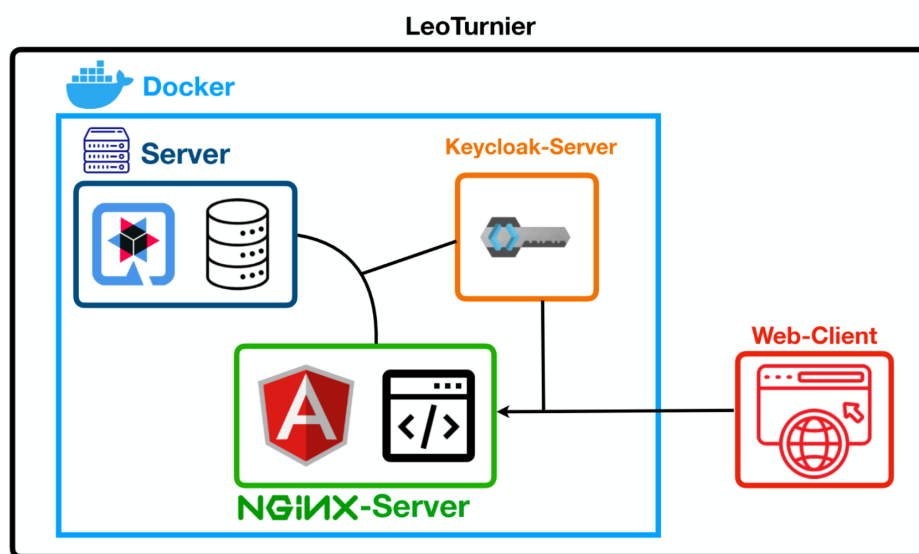


Abbildung 1: System Architecture

Die Diplomarbeit ist Aufgeteilt in Frontend (Angular), Backend (Quarkus) und einem KeyCloak Server.

### 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 greift die Quarkus Applikation mithilfe von "Quarkus OIDC" auf einen KeyCloak Server zu. Die Quarkus Applikation sowie die PostgreSQL Datenbank 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.

### 3.1.3 KeyCloak

Der Keycloak Server ist dafür verantwortlich, die Userdaten zu verwalten, und bestimmte Funktionen der Applikation nur für bestimmte Rollen freizugeben. Die Userdaten sind in einer PostgreSQL Datenbank gespeichert. Der keyCloak Server und die Datenbank werden beide jeweils in einem Docker Container ausgeführt.

Nun, da die einzelnen Komponenten der Applikation erläutert wurden, geht es weiter mit den verwendeten Technologien.

## 4 Technologien

Hier werden die einzelnen Technologien beschrieben, die für die Applikation verwendet wurden, zuerst Softwarekomponenten, danach die Entwicklungsinfrastruktur.

### 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 im Backend 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, 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, und es ist die Schnittstelle zwischen dem Frontend und Backend der Applikation. 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

Hibernate ORM ist die Schnittstelle zwischen Backend und Datenbank. 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]

Object Relational Mapping (ORM) ist eine Programmier Technik zur Konvertierung von Daten zwischen relationalen Datenbanken und objektorientierten Programmiersprachen. Mit der ORM-Technik können Objekte auf die Strukturen einer relationalen Datenbank abgebildet werden. [6]

### 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. [7]

## 4.3 PostgreSQL

Die Datenbank der Wahl ist in dieser Diplomarbeit die PostgreSQL Datenbank, da die Diplomanden 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. [8]

### 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. [8]

### 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. [8]

### 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. [8]

## 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. [9]

### 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. [10]

### 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. [10]

### 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. [10]

### 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

[10]

### 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. [10]

## 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. [11]

### 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). [11]

#### **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. [11]

### **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). [11]

### **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. [11]

## **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. [11]

## 4.6 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.

### 4.6.1 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. [12]

### 4.6.2 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. [13]

### 4.6.3 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. [14]

## 4.7 Entwicklungsinfrastruktur

Hier werden alle Technologien beschrieben, die zwar nicht Teil der Applikation sind, jedoch zum entwickeln der Applikation benutzt wurden.

### 4.7.1 Git

Als Versionskontrollsystem wird von den Diplomanden 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". [15]

Es ist ein Open-Source Projekt, das ursprünglich 2005 vom 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. [16]

#### 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: [16]

- lokale Versionsverwaltung (Local Version Control System - LVCS)

Mit dieser Architektur werden Dateien lokal einfach nur in ein separates Verzeichnis kopiert. [16]

- 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. [16]

- 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. [16]

### **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. [16]

### **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. [15]

## **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. [15]

## **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. [15]

## **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. [15]

### **4.7.2 Github**

Um gemeinsam agil an der Applikation arbeiten zu können, wird von den Diplomanden 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. [17]

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. [18]

### **Github Actions**

GitHub Actions ist ein in Github integriertes Tool, um Prozesse in einem Softwareprojekt zu automatisieren, und wird von den Diplomanden 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. [19]

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. [20]

### 4.7.3 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. [21]

#### 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: [22]

- 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. [22]

- 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. [22]

- Debugger

Ein Programm zur Prüfung anderer Programme, mit dem sich die Position von Bugs im Originalcode grafisch anzeigen lässt. [22]

## 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. [21]

## 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. [21]

## 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. [21]

### 4.7.4 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. Besitzt man jedoch die Ultimate Version von IntelliJ macht der Gebrauch von Webstorm keinen Sinn, da Ultimate alle oben genannten Features bereitstellt. [23] [24]

## 4.8 Postman

Um die Funktionalität des Quarkus Backends zu Testen, wird von den Diplomanden Postman verwendet. Haupteinsatzgebiet von Postman ist das Testen von REST APIs auf HTTP-Basis. Der grundlegende Aufbau fokussiert sich dabei auf das Verarbeiten und Validieren von Requests und deren Responses. HTTP-Requests lassen sich in


Postman mit zugehörigen Parametern über den visuellen Request-Builder spezifizieren und absenden. Die Responses können direkt angesehen und ausgewertet werden, sei es grafisch oder über die interne Programmierschnittstelle von Postman mittels JavaScript. [25]

## 4.9 MockLab

MockLab ist ein Service der eine API "mockt" sprich einen Mockup zu deutsch eine Attrappe 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. [26]

### 4.9.1 Tutorial

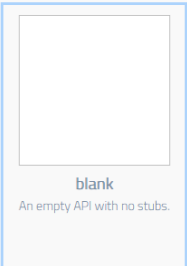
Um mit MockLab zu starten muss zuerst ein Account erstellt werden, oder man loggt sich mit seinem Github Account ein. Einmal eingeloggt werden schon alle erstellten Mock API's angezeigt bzw. beim ersten mal nur die Example Mock API.

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.



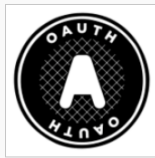
**mocklab/rest-example**  
Example mock REST API



**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

**Fußball**

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

**Fußballmannschaft**

 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



Da jetzt nun alle Bestandteile und Technologie der App geklärt sind kommt jetzt das Benutzerhandbuch.

## 5 Benutzerhandbuch

In diesem Kapitel werden alle Funktionalität der App erklärt und wer darauf Zugriff hat. Das Kapitel ist so gegliedert, dass es beim durchlesen den gesamten Ablauf der Turniererstellung, wie Durchführung erklärt, aber auch immer als Nachschlaghilfe für einzelne Pages dient.

### 5.1 Home-Page - Rollen:Alle

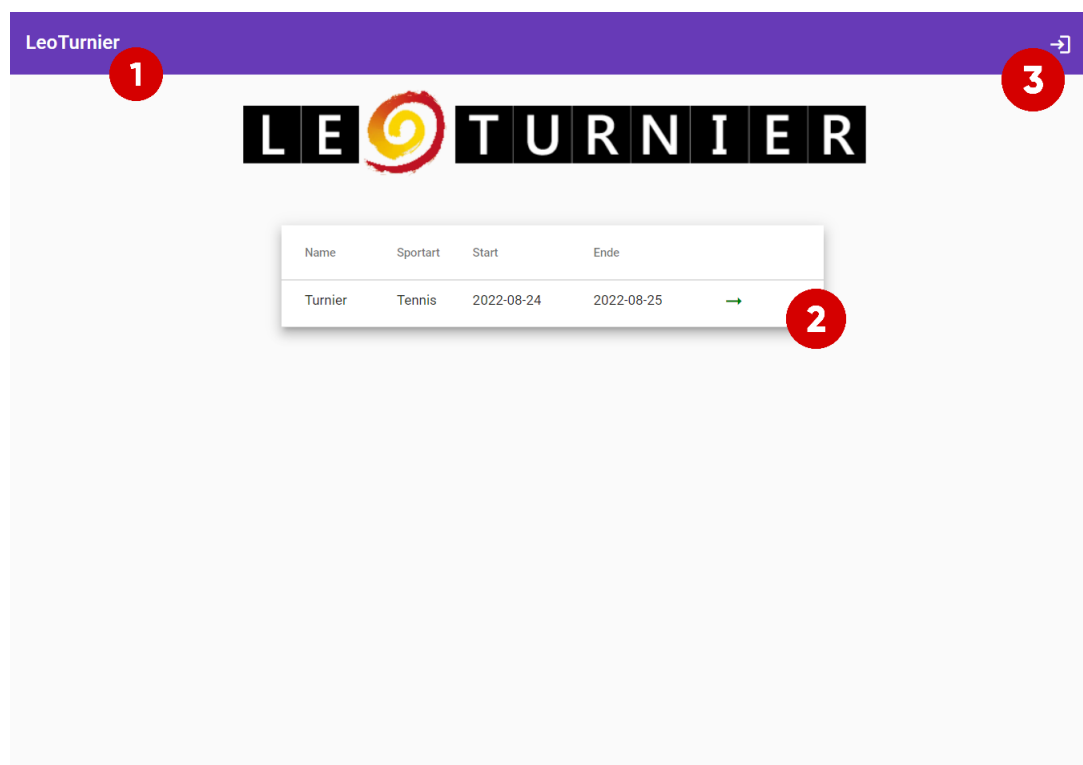


Abbildung 3: Homepage

#### 1 Titel

Links oben wird dauerhaft der Titel LeoTurnier gezeigt dieser dient auch gleichzeitig als Homebutton, also kommt man per Knopfdruck immer wieder auf diese Seite.

#### 2 Laufenden Turniere

In dieser Tabelle werden alle laufenden Turniere angezeigt. Zu jedem Turnier werden die Sportart, das Startdatum und das Enddatum angezeigt, sollten diese eingetragen sein. (*Nur der Name und der Modus sind zur Erstellung nötig*)

Mit dem rechten grünen Pfeil kommen Sie zur Tournament-View-Page(5.2) des jeweiligen Turniers.

## Login Button

Um Turniere zu erstellen und verwalten zu können ist es nötig sich vorher einzuloggen. Mit diesem Button sollten Sie direkt zum Keycloak weitergeleitet werden um sich mit ihrem Username und Password zu authorisieren. Danach werden Sie zur Login-Page(5.3) weitergeleitet werden.

## 5.2 Tournament-View-Page - Rollen: Alle

Es gibt 2 Arten von Tournament-View-Pages:

- Tree-View (für den Elimination Modus)
- Table-View (für den Round Robin Modus)

### 5.2.1 Tree-View

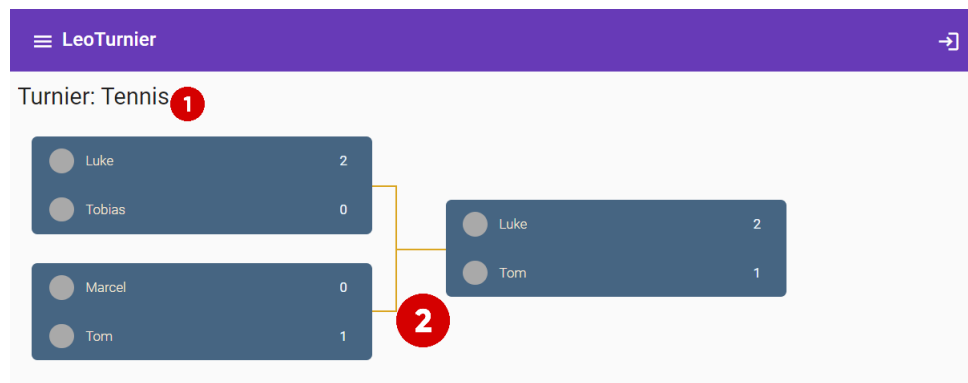


Abbildung 4: Tree-View

#### 1 Info

Hier finden Sie eine kurze Info zum Turniernamen und Sportart




#### 2 Turnierbaum

Hier finden Sie den Turnierbaum, dieser zeigt alle Ergebnisse bzw. aktuellen Spielstände eines Turnieres an. Im Screenshot sehen Sie eine der einfachsten Formen eines Turnierbaums mit 4 Spielern. Sollte der Turnierbaum bei steigender Spieleranzahl zu groß werden um alles auf einen Bildschirm zu sehen verwenden Sie bitte den Slider am unterem Bildschirmrand.

Das Neuladen des Turniers funktioniert auf dieser Seite nicht. Um den aktuellsten Stand des Turniers zu bekommen navigieren Sie zurück auf die Home-Page und wieder auf den grünen Pfeil.

### 5.2.2 Table-View

Turnier-RR: Fußball **1**

Competitor	Placement
Luke	
Marcel	
Tom	
Tobias	4 <b>2</b>

Spiele:

Competitor 1	Score	Competitor 2
Luke	2 : 0	Tobias
Marcel	2 : 0	Tom
Luke	2 : 0	Marcel
Tobias	0 : 1	Tom
Luke	1 : 0	Tom
Tobias	0 : 2	Marcel

**3**

Abbildung 5: Table-View

#### **1** Info

Hier finden Sie eine kurze Info zum Turniernamen und Sportart

#### **2** Platzierungen

In dieser Tabelle werden alle Spieler, nach Placement aufsteigend sortiert, angezeigt.

#### **3** Spiele

In dieser Tabelle werden alle Ergebnisse bzw. aktuellen Spielstände eines Turnieres angezeigt. Die ersten drei Plätze erhalten dabei auch eine Medaille.

## 5.3 Login-Page - Rollen: Admin, Tournament-Organizer

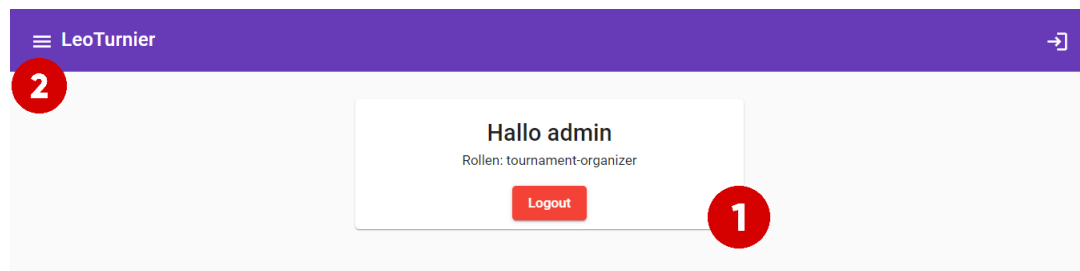


Abbildung 6: Login-Page

### 1 Rollen und Logout Button

Hier finden Sie alle Ihnen zugewiesenen Rollen sowie einen Logout-Button der ihre Session beendet und Sie zurück zur Home-Page leitet.

### 2 Navigation-Burger

Nachdem Sie jetzt eingeloggt sind erscheint oben rechts ein Nav-Burger. Per Knopfdruck zeigt dieser Ihnen nun eine Liste aller Pages die zur Turnierverwaltung nötig sind.

## 5.4 Player-Page - Rollen:Admin,Tournament-Organizer

Id	Name	Geburtsdatum	Team
1	Luke	2000-07-05	
2	Tom	2002-08-15	
3	Marcel	2001-05-07	
4	Tobias	2003-03-13	

Abbildung 7: Player-Overview-Page

### 1 Create-Button

Button zum Erstellen neuer Spieler und leitet Sie zur Player-Details-Page(5.4.1).

### 2 Players

In dieser Tabelle werden alle Spieler mit ihren Details angezeigt. Zu jedem Spieler werden hier seine Id, Name, Geburtsdatum und zugehöriges Team angezeigt, sollten dieser vorhanden sein. (*Nur der Name ist zur Erstellung nötig*)

### 3 Actions

Diese Spalte beinhaltet zwei Icon-Buttons:

- Löschen eines Spielers (Sie müssen diese Aktion zwei mal bestätigen)
- Updaten eines Spielers(Sie werden danach zur Player-Details-Page(5.4.1)geleitet)

#### 5.4.1 Player-Details-Page

Name

Pflichtfeld!!!

Geburtsdatum

Submit

Abbildung 8: Player-Details-Page



Diese Page ist zum Erstellen und Updaten von Spielern. Die Mindestanforderungen sind hierbei nur ein Name.

## 5.5 Team-Page - Rollen:Admin,Tournament-Organizer

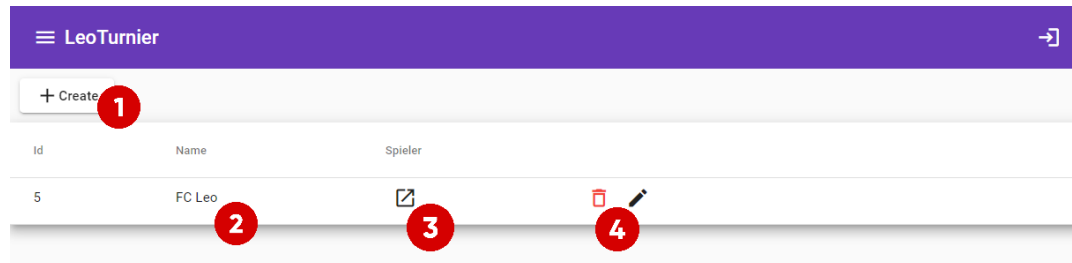


Abbildung 9: Team-Overview-Page

### 1 Create-Button

Button zum Erstellen neuer Teams und leitet Sie zur Team-Details-Page(5.5.1).

### 2 Teams

In dieser Tabelle werden alle Teams mit ihren Details angezeigt. Zu jedem Team werden hier seine Id, Name und zugehörige Spieler angezeigt, sollten dieser vorhanden sein. *(Nur der Name ist zur Erstellung nötig)*

### 3 Spieler

In der Spalte Spieler finden Sie einen Icon-Button zu einem PopUp-Fenster welches eine Liste aller Spieler des Teams anzeigt.

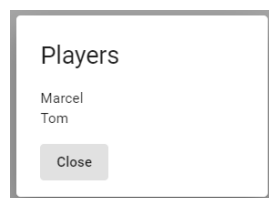




Abbildung 10: Team-Players-PopUp

### 4 Actions

Diese Spalte beinhaltet zwei Icon-Buttons:

-  Löschen eines Teams (Sie müssen diese Aktion zwei mal bestätigen)
-  Updaten eines Teams (Sie werden danach zur Team-Details-Page (5.5.1) geleitet)

### 5.5.1 Team-Details-Page

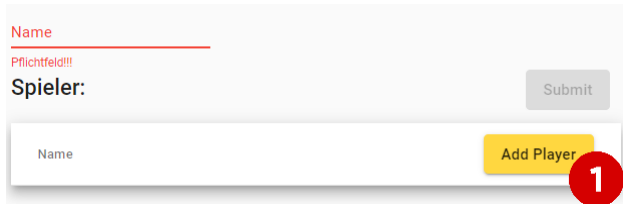


Abbildung 11: Team-Details-Page

Diese Page ist zum Erstellen und Updaten von Spielern. Die Mindestanforderungen sind hierbei nur ein Name.

#### **1** Add-Player

Bei Knopfdruck des Add-Player Button erscheint eine Liste aller Spieler die nicht in dem Team sind. (*Keine mehrfach Auswahl*)

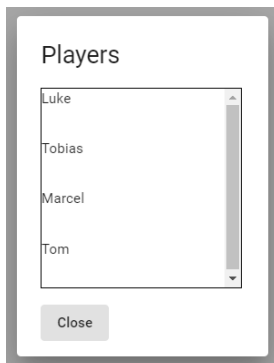


Abbildung 12: Add-Player-PopUp (Team)

## 5.6 Tournament-Page

### Rollen:Admin,Tournament-Organizer

Id	Name	Start	Ende	Sportart	Modus
1	Turnier	2022-08-24	2022-08-25	Tennis	Elimination
2	Turnier-RR	2022-08-25	2022-08-26	Fußball	Round Robin

Abbildung 13: Tournament-Overview-Page

#### 1 Create-Button

Button zum Erstellen neuer Tournaments und leitet Sie zur Tournament-Details-Page(5.6.1).

#### 2 Tournaments

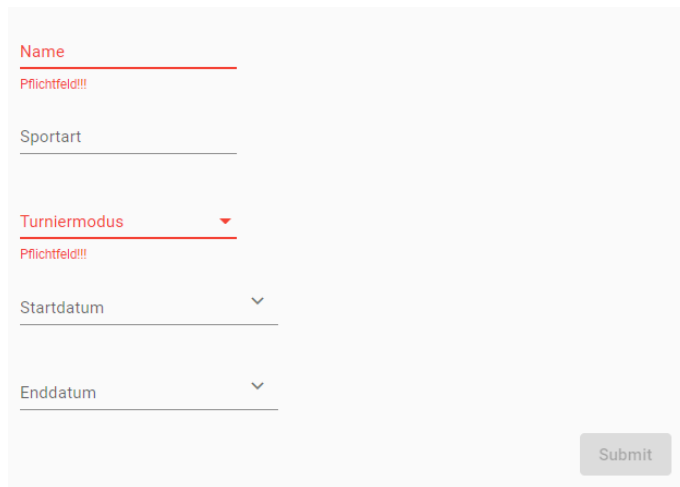
In dieser Tabelle werden alle Tournaments mit ihren Details angezeigt. Zu jedem Tournament werden hier seine Id, Name, Startdatum, Enddatum, Sportart und Turniermodus angezeigt, sollten dieser vorhanden sein. (*Nur der Name und der Modus sind zur Erstellung nötig*)

#### 3 Actions

Diese Spalte beinhaltet zwei bis drei Icon-Buttons:

- Löschen eines Tournaments (Sie müssen diese Aktion zwei mal bestätigen)
- Updaten eines Tournaments (Sie werden danach zur Tournaments-Details-Page(5.6.1) geleitet)
- Starten eines Tournaments (Sie müssen diese Aktion zwei mal bestätigen)
- Leitet Sie zur Match-Overview-Page(5.7) des Tournaments

### 5.6.1 Tournament-Details-Page



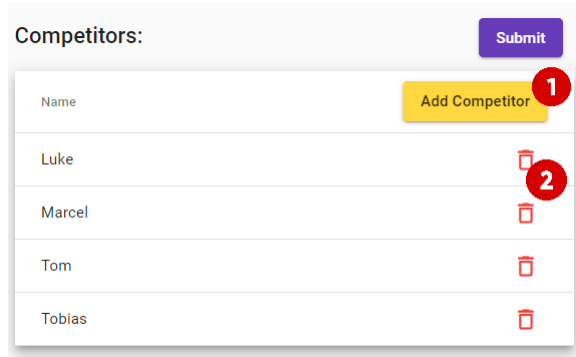
The screenshot shows a form for creating a tournament. It includes the following fields and elements:

- Name**: A text input field with a red underline and the label "Pflichtfeld!!!".
- Sportart**: A text input field.
- Turniermodus**: A dropdown menu with a red underline and the label "Pflichtfeld!!!".
- Startdatum**: A date input field with a dropdown arrow.
- Enddatum**: A date input field with a dropdown arrow.
- Submit**: A grey button at the bottom right.

Abbildung 14: Tournament-Details-Page

Diese Page ist zum Erstellen von Turnieren. Die Mindestanforderungen sind hierbei nur ein Name und der Turniermodus.

Nachdem Sie ein Turnier erstellt haben erweitert sich die Tournament-Details-Page um die Funktion, Competitor(Spieler, Teams) für ein Turnier einzutragen.



The screenshot shows the "Competitors:" section of the tournament details page. It includes a purple "Submit" button and a yellow "Add Competitor" button (marked with a red circle 1). Below the "Add Competitor" button is a table listing competitors:

Name
Luke
Marcel
Tom
Tobias

Each row in the table has a red trash icon (marked with a red circle 2) to the right of the name.

Abbildung 15: Tournament-Details-Page mit Add-Player Funktion

#### 1 Add Competitor

Bei Knopfdruck des Add-Competitor Button erscheint eine Liste aller Spieler die nicht in dem Team sind. (*Keine mehrfach Auswahl*)

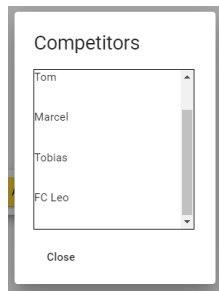


Abbildung 16: Add-Player-PopUp (Tournament)

## 2 Actions

Mit dem Icon-Button  löschen Sie die Teilnahme eines Competitors für das Turnier.  
(*Nicht den Competitor*)

## 5.7 Match-Overview-Page - Rollen:Admin,Tournament-Organizer

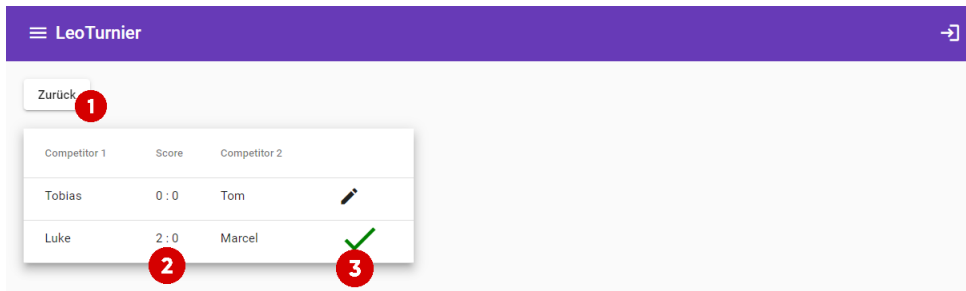


Abbildung 17: Match-Overview-Page

### 1 Zurück-Button



Dieser Button leitet Sie zurück zur Tournament-Overview-Page.

### 2 Spiele

In dieser Tabelle werden alle Matches mit ihren beiden kontrahierenden Competitors und Spielstand angezeigt.

### 3 Actions

Diese Spalte beinhaltet einen Icon-Button oder ein Icon:

-  Updaten eines eines Matches (Sie werden danach zur Match-Submission-Page(5.7.1) geleitet)
-  Zeigt an dass, ein Spiel fertig gespielt ist .

### 5.7.1 Match-Submission-Page

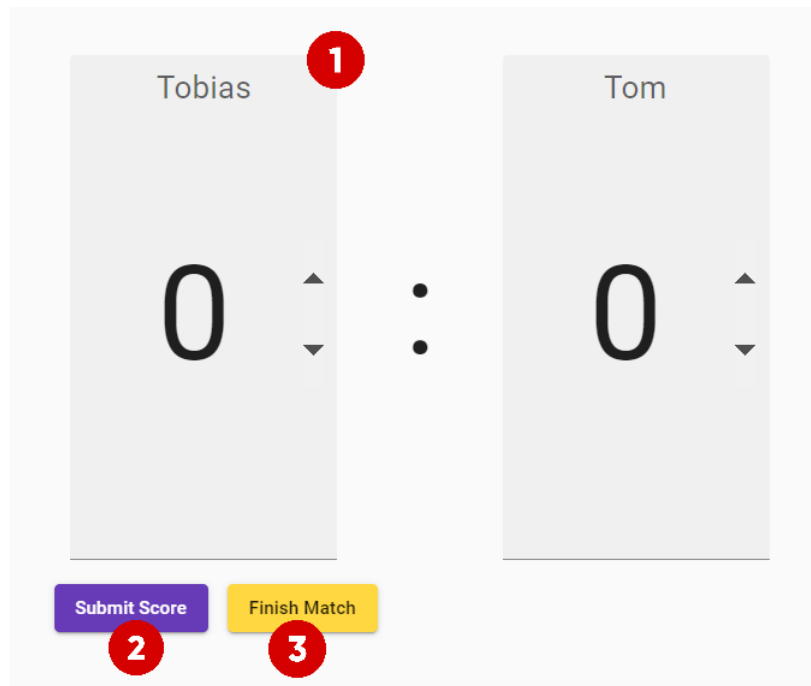


Abbildung 18: Match-Submission-Page

#### 1 Score-Tile

Die Match-Submission-Page enthält zwei Match-Tiles mit denen man den Score eines Spiels eintragen kann. Der Punktestand kann entweder über eine Tastatureingabe, oder über die zwei Pfeile rechts neben dem Punktestand eingegeben werden. Dabei muss der Spielstand eine Zahl zwischen 0-999 betragen.

*(Mit einer so großen Spanne sollen mehr Sportarten möglich sein, da sich Punktestände von Sportart zu Sportart sehr stark ändern können z.B: Basketball (113-72) und Fußball(2-1))*

#### 2 Submit-Score-Button

Bei Knopfdruck von Submit-Score wird der Punktestand geupdated und Sie werden zur Match-Overview-Page geleitet.

#### 3 Finish-Match-Button

Bei Knopfdruck von Finish-Match wird der Punktestand geupdated und das Match beendet. Danach werden Sie zur Match-Overview-Page geleitet.

Da nun die Bedienung der App klar ist wird nun erklärt wie dabei das Frontend umgesetzt wurde.



# 6 Umsetzung Frontend

Hier werden alle Funktionalitäten des Frontends erklärt und wie es mithilfe der verschiedenen Libraries arbeitet. Anfangs wird die Ordnerstruktur erklärt um sich erstmal in einer Angular App zurechtzufinden.

## 6.1 Ordnerstruktur

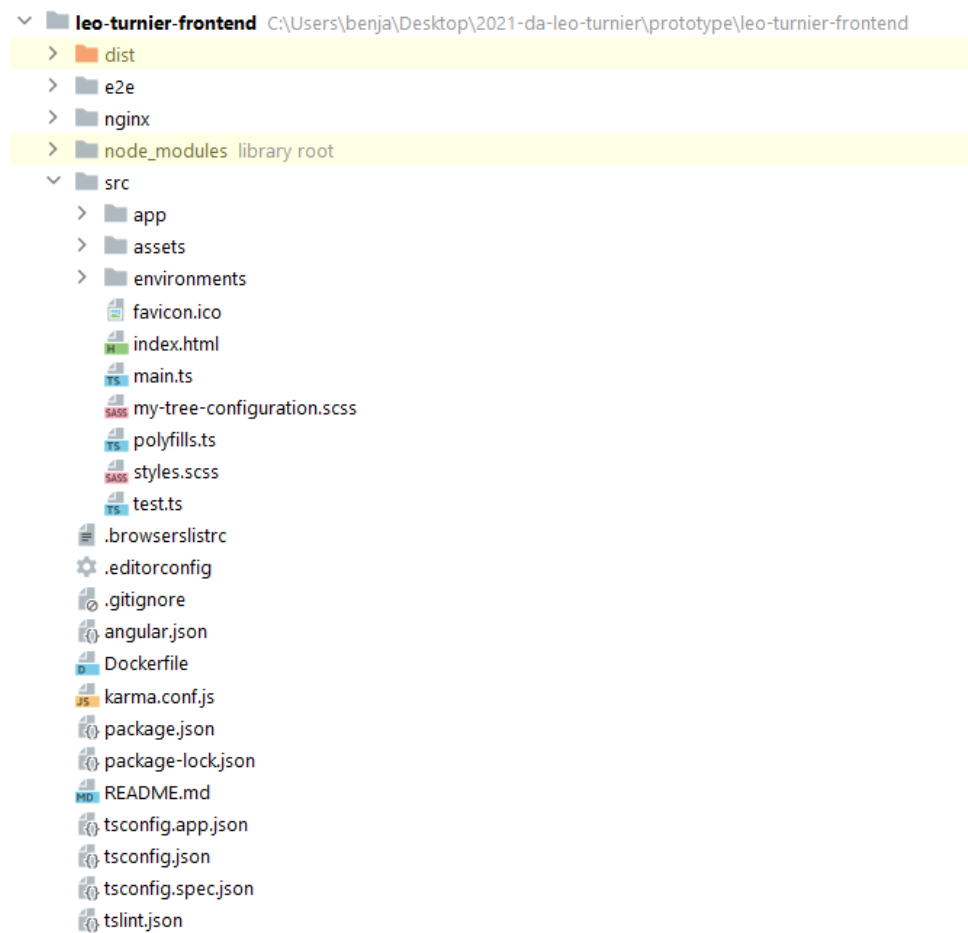


Abbildung 19: Angular Ordnerstruktur

Bei der Erstellung einer Angular Applikation wird schon fast die gesamte Ordnerstruktur festgelegt. Bis auf ein paar Ausnahmen wie der Nginx Ordner, oder die zwei in Gelb markierten Ordner, welche nach dem dem Laden bzw. Builden generiert werden, hat sich hier nichts getan. Wie an den meisten Filenamen schon zu erkennen ist befinden

sich auf dieser Ebene so fast nur Files, die zur Konfiguration nötig sind. Das wichtigste File dabei ist "package.json".

```
{
  "name": "leo-turnier-frontend",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~12.2.16",
    "@angular/cdk": "^12.2.13",
    "@angular/common": "~12.2.16",
    "@angular/compiler": "~12.2.16",
    "@angular/core": "~12.2.16",
    "@angular/forms": "~12.2.16",
    "@angular/localize": "~12.2.16",
    "@angular/material": "^12.2.13",
    "@angular/platform-browser": "~12.2.16",
    "@angular/platform-browser-dynamic": "~12.2.16",
    "@angular/router": "~12.2.16",
    "@ng-bootstrap/ng-bootstrap": "^9.1.3",
    "bootstrap": "^4.5.0",
    "keycloak-angular": "^10.0.0",
    "keycloak-js": "^18.0.1",
    "ng-tournament-tree": "^2.0.2",
    "rxjs": "~6.6.0",
    "tslib": "^2.0.0",
    "zone.js": "~0.11.4"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "^12.2.18",
```

Abbildung 20: package.json

Hier werden alle Metadaten, Scripts und Dependencies der App aufgelistet. Bei den Dependencies unterscheidet man zwischen Dependencies und DevDependencies.

- Dependencies werden transitiv installiert: Wenn A B benötigt und B C benötigt, wird C installiert, sonst könnte B nicht funktionieren und A auch nicht.
- DevDependencies werden nicht transitiv installiert. Wir brauchen z.B. B nicht zu testen, um A zu testen, also können die Testabhängigkeiten von B weggelassen werden.

Um diese Dependencies auch nutzen zu können muss bei jedem Angular Projekt "npm install" ausgeführt werden um diese zu laden. Danach werden sie dann im Ordner node\_modules gespeichert. Dieser sollte, aber vor jeder Weitergabe gelöscht werden, oder wie zum Beispiel in Git im ".gitignore" inkludiert werden, da dieser mit Abstand am speicherintensivsten ist.

Auch wenn Angular eine große Anzahl an Ordnern und Files bietet geschieht wahrscheinlich 99% der Arbeit eines Angular-Projekt im app Ordner. Dieser enthält alle Components sowie Klassen die für die Applikation benötigt werden.

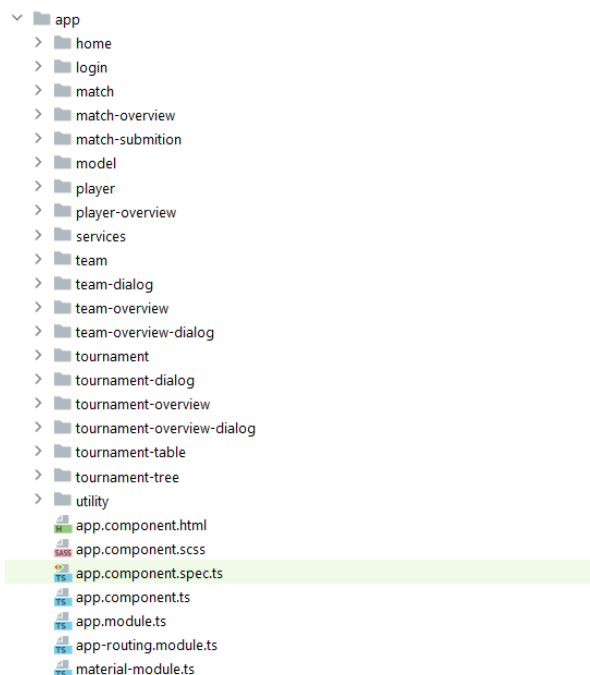


Abbildung 21: App Ordner

Um ein bisschen Ordnung ins Chaos zu bringen kann man diese Files bzw Ordner in drei Kategorien unterteilen:

- App Components: Alle Files die mit app anfangen
- Hilfsklassen: model, services, utility, material-module.ts
- Components: Alle restlichen Ordner.

### 6.1.1 App Components

#### Modules und Routing

Modules und Routing sind der Kern einer Angular App. Hier laufen alle Components zusammen und Libraries werden über Imports eingebunden. Im Routing findet unter routes ein Array mit allen paths, sowie deren Components. Mit @NgModule wird das App-Modul erstellt. Dort findet man dann auch alle Abhängigkeiten und Komponenten, die dieses Modul enthält. Um einen guten Überblick über alle Modules zu haben, wurden für die große Anzahl an AngularMaterial Modules ein eigens Module erstellt. Beim Laden dieses Moduls wird der AppComponent gestartet. [27]

```
const routes: Routes = [
  { path: '', pathMatch: 'full', redirectTo: 'home' },
  { path: 'home', component: HomeComponent },
  { path: 'tournaments', component: TournamentOverviewComponent },
  { path: 'tournaments/:id', component: TournamentComponent },
  { path: 'tournament-tree', component: TournamentTreeComponent },
  { path: 'tournament-table/:id', component: TournamentTableComponent },
  { path: 'players', component: PlayerOverviewComponent },
  { path: 'players/:id', component: PlayerComponent },
  { path: 'team', component: TeamOverviewComponent },
  { path: 'team/:id', component: TeamComponent },
  { path: 'matches/:id', component: MatchOverviewComponent },
  { path: 'match-submission/:id/:tournamentId', component: MatchSubmissionComponent },
  { path: 'login', component: LoginComponent, canActivate: [AuthGuard] }
];
```

Abbildung 22: routes in app-routing.module.ts

```
@NgModule({
  declarations: [
    AppComponent,
    TournamentOverviewComponent,
    LoginComponent,
    PlayerOverviewComponent,
    TeamOverviewComponent,
    TeamOverviewDialogComponent,
    PlayerComponent,
    TeamComponent,
    TeamDialogComponent,
    TournamentComponent,
    HomeComponent,
    TournamentTreeComponent,
    MatchComponent,
    TournamentDialogComponent,
    MatchOverviewComponent,
    MatchSubmissionComponent,
    TournamentOverviewDialogComponent,
    TournamentTableComponent,
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserModuleAnimationsModule,
    MaterialModule,
  ],
})
```

Abbildung 23: @NgModule in app.module.ts

## App-Component

Die App Component ist der "Vater" jeder Angular App. Sie enthält alle anderen Child-Components und ist jederzeit sichtbar. Daher eignet Sie sich perfekt für eine Tool- und Navbar. Wie alle Component besteht Sie aus 4 Files:

- TypeScript (.ts): Ist der Code-Behind und enthält die gesamte Business Logik der Component.
- HTML
- SCSS: Durch SCSS wird die Schreibweise von CSS vereinfacht und Variablen fest definiert.
- spec.ts: Die spec-Dateien sind Unit-Tests für den Sourcecode. Die Konvention für Angular-Anwendungen ist es, eine .spec.ts-Datei für jede .ts-Datei zu haben. [28]

## 6.1.2 Hilfsklassen

### Model

Hier sind alle Klassen der App untergebracht. Bis auf ein paar Abweichung findet man hier das gleiche Datenmodel wie im Backend(7.1) wieder. Da Angular mit Typescript verwendet wird, können für diese Klasse type checking verwendet werden (in vielen Fällen können stattdessen auch ein Interface verwendet werden)

```
import {Competitor} from './competitor.model';
import {Team} from './team.model';

export class Player extends Competitor{
  constructor(public id: number,
              public name: string,
              public birthdate: Date,
              public team: Team) {
    super(id, name);
  }
}
```

Abbildung 24: player.model.ts

### Services

Components sollten Daten nicht direkt abrufen oder speichern. Sie sollten sich auf die Darstellung von Daten konzentrieren und den Datenzugriff an einen Service delegieren. In dem Ordner services befindet sich daher der gesamte Service, aufgeteilt in Aufgabenbereiche. Jeder Service ist dabei gleich aufgebaut. Damit Angular den Service in eine Component injecten kann muss vorher ein Provider eingetragen werden. Ein Provider ist etwas, das einen Service erstellen oder bereitstellen kann. Um sicherzustellen, dass der Service diesen anbieten kann, registrieren Sie ihn beim Injektor. Der Injektor ist das Objekt, das den Provider auswählt und dort injiziert, wo die Anwendung ihn benötigt. Standardmäßig trägt Angular hierfür 'root' ein. Wenn ein Dienst auf Root-Ebene bereitgestellt wird, erstellt Angular eine einzelne, gemeinsam genutzte Instanz des Services und injiziert diese in jede Klasse, die danach fragt. [29]

Um Daten auf die Daten des Backends zugreifen zu können wird die Client-HTTP-API verwendet, welcher zugleich noch Features für Error-Handling und Typed Response Objects bereitstellt. [30]

Da nicht jeder Nutzer von LeoTurnier Zugriff zu allen Daten haben darf, wird dazu noch ein Keycloak-Service verwendet der für jeden Request einen Authentication-Token zur

Verfügung stellt. Damit dieser Service funktioniert muss man zuerst den Keycloak-Server mit dem Frontend verbinden.

```
@Injectable({
  providedIn: 'root'
})
export class TournamentService {
  private host = 'http://localhost:8080/api/';

  constructor(private httpClient: HttpClient, public datePipe: DatePipe, private keycloakService: KeycloakService) {
  }
}
```

Abbildung 25: tournament.service.ts

```
async startTournament(id: string): Promise<void> {
  const authToken = this.keycloakService.getToken();
  const headers = {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${authToken}`
  };
  await this.httpClient.get( url: this.host + 'execution/startTournament?tournamentId=' + id, options: {headers}).subscribe(
    next: data => console.log('success', data),
    error: error => console.log('oops', error)
  );
}
```

Abbildung 26: Beispiel Request: startTournament

## Utility - Keycloak

In diesem Ordner befinden sich alle nötigen Files für eine funktionierende Verbindung mit Keycloak. Für die Initialisierung des KeycloakService ist eine Methode im "app.init.ts" zuständig. Als Vorlage diente hierfür das Tutorial des Keycloak-Angular Module von Mauricio Vigolo [1]

```
import {KeycloakService} from 'keycloak-angular';

export function initializeKeycloak(keycloak: KeycloakService): () => Promise<boolean> {
  return () =>
    keycloak.init( options: {
      config: {
        url: 'http://localhost:8443',
        realm: 'leoturnier',
        clientId: 'angular-leoturnier-client'
      },
      initOptions: {
        checkLoginIframe: false,
        checkLoginIframeInterval: 25,
        onLoad: 'check-sso'
      },
      loadUserProfileAtStartUp: true
    });
}
```

Abbildung 27: app.init.ts

Wie im Screenshot zusehen befindet sich hier eine Konfiguration für einen Client von Keycloak. Dieser kann dann an Keycloak eine Anforderung stellen, um einen Benutzer zu

authentifizieren, Identitätsinformationen einzuholen, oder einen Zugriffstoken anzufordern. Der Client muss direkt im Keycloak mit der Administration Console erstellt werden und muss mit der Konfiguration und der Frontend URL genau übereinstimmen.

**angular-leoturnier-client** OpenID Connect

Clients are applications and services that can request authentication of a user.

Settings Roles Client scopes Sessions Advanced

### General Settings

Client ID \* ⓘ angular-leoturnier-client

Name ⓘ

Description ⓘ

Always display in console ⓘ ☐ Off

### Access settings

Root URL ⓘ http://localhost:8443

Home URL ⓘ

Valid redirect URIs ⓘ http://localhost:4200/\* + Add valid redirect URIs

Valid post logout redirect URIs ⓘ http://localhost:4200/\* + Add valid post logout redirect URIs

Web origins ⓘ http://localhost:4200 + Add web origins

Admin URL ⓘ http://localhost:4200

Abbildung 28: Keycloak-Frontend Client

### 6.1.3 Components

Components sind die UI-Bausteine einer jeden Angular App. Wie vorher schon erwähnt sind alle anderen Komponenten Child-Components der App-Component und bilden somit Component-Tree. LeoTurnier ist hierbei sehr simpel gehalten und besitzt einen fast komplett flachen Baum. Auch wenn jede Component einzigartig ist und ihren eigenen Funktionalitäten hat funktionieren haben sie viele Gemeinsamkeiten:

## Service

Um beispielsweise eine Tabelle ein Formular zu füllen braucht jede Component mindestens einen Service. Dieser wird im Konstruktor injiziert, aber kann in ihm noch nicht verwendet werden. In den meisten Fällen wie bei einer Tabelle müssen die Daten aber schon zu Beginn bereitgestellt werden. Hier kommt die Methode `ngOnInit` ins Spiel.

## OnInit

Um `ngOnInit` verwenden zu können muss vorher das die Component vorher `OnInit` implementieren. Im Grunde handelt es sich hierbei nur um eine Lifecycle-Hook, welche Angular selbst verwaltet. Die Methode wird dann nur zur Initialisierung der Component aufgerufen, wie uns der Name schon verrät. Somit wird sofort nach dem Konstruktor die Methode aufgerufen. Dabei können Daten schon vor dem Start der Component beschaffen werden und danach vielleicht Tabelle befüllt werden. [31]

```
async ngOnInit(): Promise<void> {  
  this.dataSource.data = await this.api.getPlayers();  
}
```

Abbildung 29: Beispiel Router-Call aus `player-overview.component.ts`

## Router

Will man jetzt in dieser Tabelle die beispielsweise mit Spielern einen bearbeiten, ist eine eigene Component nötig. Um es also für eine Component zu ermöglichen auf eine andere Component zu navigieren, ist ein sogenannter Router nötig. Dieser wird genau wie der Service injiziert und gehört zum RouterModule. Nun kann der Router mithilfe der Routes im `"app-routing.module.ts"` den Benutzer zu einer anderen Component navigieren und für diese auch Parameter mitgeben.

```
this.router.navigate( commands: ['player-detail'], extras: {state: {data: id}});
```

Abbildung 30: Beispiel Router-Call aus `player-overview.component.ts`

Diese Parameter müssen dann auch von der anderen Component gespeichert werden. Hier schafft der `ActivatedRouter` abhilfe. Das Prozedere ist hierbei genau gleich wie beim Router und der Aufruf kann sofort im Konstruktor stattfinden.

```
activatedRouter.paramMap.subscribe( next: map => this.id = map.get('id'));
```

Abbildung 31: Beispiel `ActivatedRouter`-Call aus `player.component.ts`



## 6.2 Tournament-Tree

Eine ansprechenden Aufgaben des Frontend ist die Visualisierung der Turniere, vor allem die des Elimination stellen eine große Herausforderung dar. Darum verwendet das Frontend für Erstellung eines optisch ansprechenden Turnierbaums das Module NgTournament was diese Aufgabe massiv erleichtern soll. [32]

Die hierfür benötigten Components finden sich im Order Match, Tournament-Tree und Home.

### Home

Zuallererst braucht NgTournament Information über den gesamten Turnierverlauf. Diese müssen in Form eines NggtTournament vorhanden sein, welches wiederum aus Rounds und Matches besteht. Den Algorithmus hierfür stellt NgTournament bereit, dieser muss nur auf das eigene Datenmodell geändert werden und schon sollte es gehen, da NgTournament aber leider nur fertig gespielte Turniere visualisieren kann, musste der Algorithmus noch so ergänzt werden, dass bei noch nicht gespielten Matches als "Ausstehend" aufgeführt werden.

```

    } else if ('competitor1' in node.match) {
      round.matches.push(
        {
          teams: [
            // @ts-ignore
            {name: node.match.competitor1.name, score: 0},
            {name: 'Ausstehend', score: 0},
          ]
        }
      );
    } else if ('competitor2' in node.match) {
      round.matches.push(
        {
          teams: [
            {name: 'Ausstehend', score: 0},
            // @ts-ignore
            {name: node.match.competitor2.name, score: 0}
          ]
        }
      );
    }
  }
}
else{
  round.matches.push(
    {
      teams: [
        {name: 'Ausstehend', score: 0},
        {name: 'Ausstehend', score: 0}
      ]
    }
  );
}
}

```

Abbildung 32: hinzufügen ausstehender Matches in showTree(Home-Component)

## Tournament-Tree & Home

Nachdem das Tournament als NggtTournament vorhanden ist wird dieses an das Tournament-Tree Component geleitet, sie komplett von NgTournament bereitgestellt und muss auch nicht mehr bearbeitet werden. Nun rendert die Tournament-Tree Component mithilfe der Match Component als Template, welche auch vollständig bereitgestellt wird, den kompletten Turnierbaum.

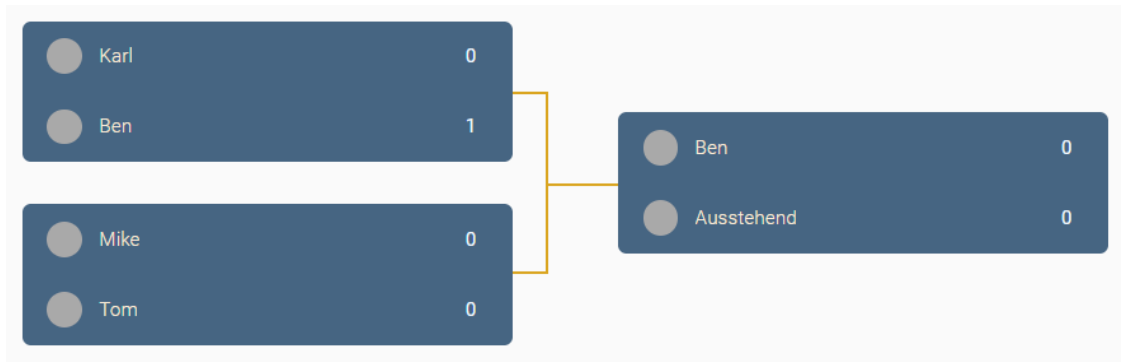


Abbildung 33: Turnierbaum

## Styling

Das Styling für den Turnierbaum wird bereits beim Laden des NgTournament hinzugefügt. Doch mit da der Baum auf weißen Hintergrund platziert ist passte es besser, wenn der Turnierbaum auch einen weißen Hintergrund besitzt. Um das grundlegende Styling zu überschreiben schafft das File "my-tree-configurations.scss" abhilfe, welches nur im "style.scss" vor "ng-tournament-tree/styles/ngtt-styles" imported werden muss.

```
$ngtt-background-color: #fafafa;
```

Abbildung 34: my-tree-configurations.scss

```
@import "my-tree-configuration";
@import '~ng-tournament-tree/styles/ngtt-styles';
```

Abbildung 35: style.scss

# 7 Umsetzung Backend

In diesem Kapitel wird erklärt, wie das Backend der Applikation implementiert wurde. Es wird zuerst das Datenmodell erläutert, dann die Ordnerstruktur von Quarkus und die verschiedenen Arten von Klassen, die darin zu finden sind, und zuletzt wird die Turnierlogik der verschiedenen Turniermodi erklärt.

## 7.1 Datenmodell

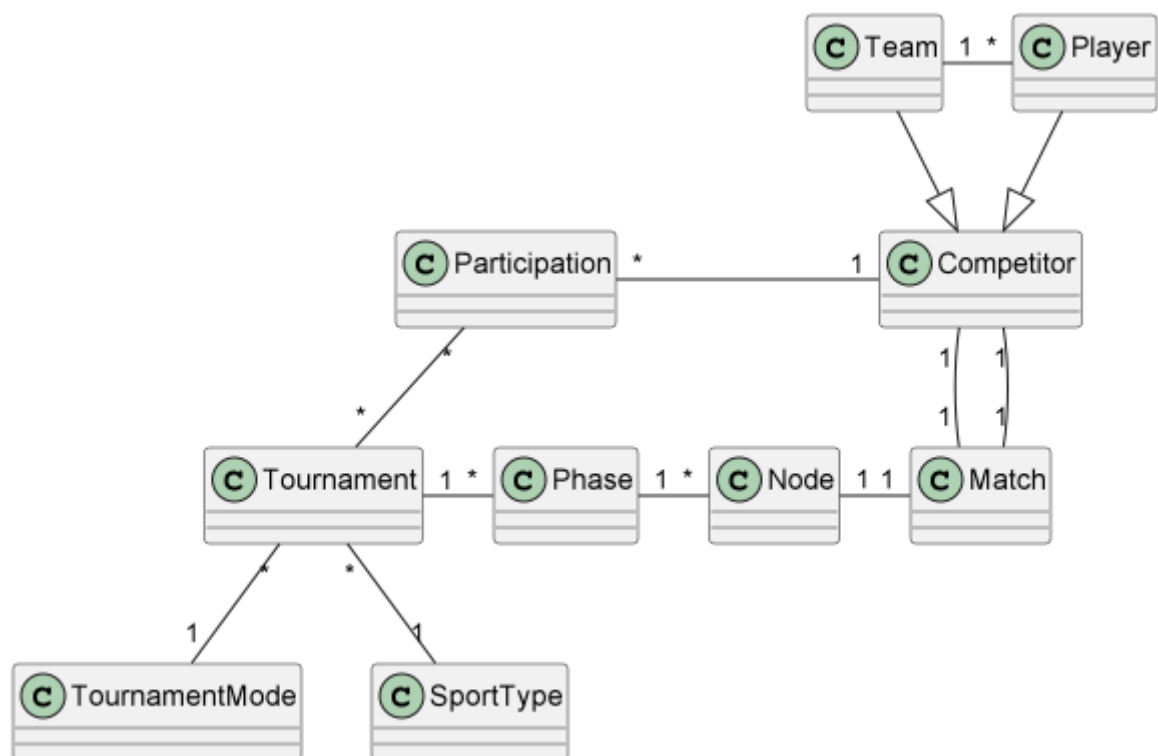


Abbildung 36: Class Diagram

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/innen (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.

## 7.2 Quarkus Backend

### 7.2.1 Ordnerstruktur

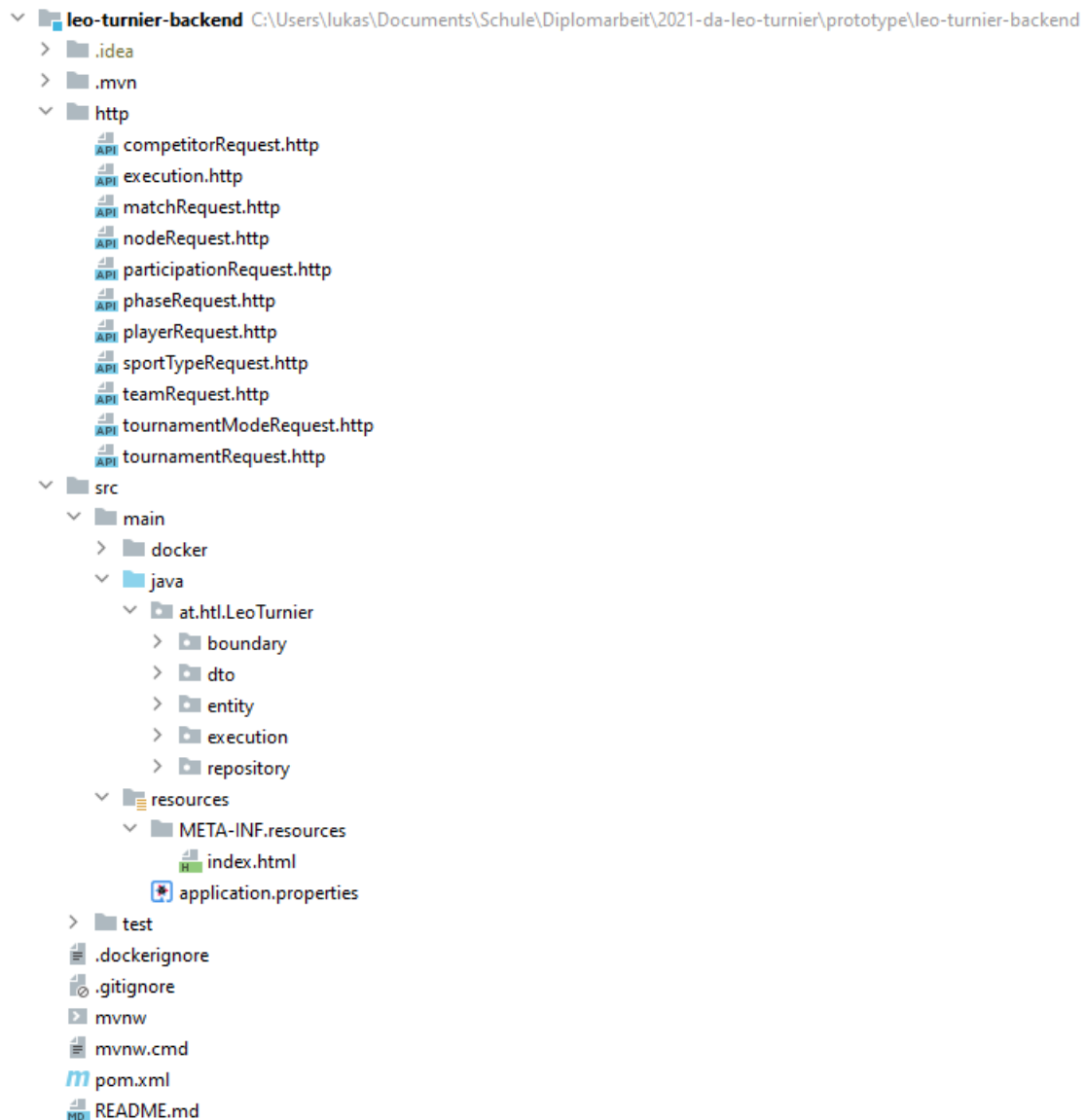


Abbildung 37: Quarkus Ordnerstruktur

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 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <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>...</dependencyManagement>
  <dependencies>...</dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>${quarkus.platform.group-id}</groupId>

```

Abbildung 38: 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 (siehe Kapitel "Deployment"), ein "java" Ordner, in dem sich die Java Klassen der Applikation befinden, 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 39: application.properties

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

Die Java Klassen im "java" Ordner sind weiters in 5 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. Als nächstes gibt es die im "execution" Ordner, diese sind für die Durchführung der Turniere zuständig, hier befindet sich auch der Großteil der Logik. 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.

## 7.2.2 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 40: 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. Wie der

Wert der Id generiert wird, wird mit "@GeneratedValue" festgelegt. 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 Assoziationstabelle 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.



### 7.2.3 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 41: 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 42: 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.

## 7.2.4 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 43: 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 44: 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 hinzugefügten, da diese für die Turnierdurchführung jeweils einzeln Implementiert werden müssen.

### 7.2.5 Turnierdurchführung

Für die Turnierdurchführung gibt es insgesamt 4 verschiedene Durchführungsklassen, eine für jeden Turniermodus (Elimination, Round Robin, Combination), und eine als Schnittstelle zu den Serviceklassen, welche aus den anderen 3 die richtigen Methoden ausführen soll, abhängig davon, welchen Turniermodus das Turnier hat.

```
public Tournament startTournament(Long tournamentId, Integer numOfGroups) {
    Tournament tournament = tournamentRepository.getById(tournamentId);
    if(tournament == null || tournament.getTournamentMode() == null) {
        return null;
    }
    clearTournament(tournamentId);
    List<Competitor> competitors = competitorRepository.getByTournamentId(tournament.getId());
    if (tournament.getTournamentMode().getName().equals("Round Robin")) {
        tournament = roundRobinExecution.startTournament(tournament, competitors, groupNumber: -1);
    } else if (tournament.getTournamentMode().getName().equals("Combination") && numOfGroups != null) {
        tournament = combinationExecution.startGroupPhase(tournament, competitors, numOfGroups);
    } else {
        tournament = eliminationExecution.startTournament(tournament, competitors);
    }
    return tournament;
}
```

Abbildung 45: "startTournament" Methode der Execution Klasse

In der "startTournament" Methode zum Beispiel wird, nachdem alle vorherigen Matches usw. daraus gelöscht wurden, die "startTournament" Methode aus der dem Turniermodus entsprechenden Durchführungsklasse ausgeführt, bzw. im Falle des "Combination" Turniermodus die "startGroupPhase" Methode (mehr dazu weiter unten).

### 7.2.6 Elimination

Im Elimination Turniermodus scheidet nach einem Match der Verlierer sofort aus dem Turnier aus, und der Gewinner steigt in die nächste Runde auf. Das Ganze wiederholt sich so lange, bis nur noch ein/eine Teilnehmer/in übrig bleibt, dieser ist dann der Sieger des Turniers. So entsteht dann der klassische Turnierbaum.

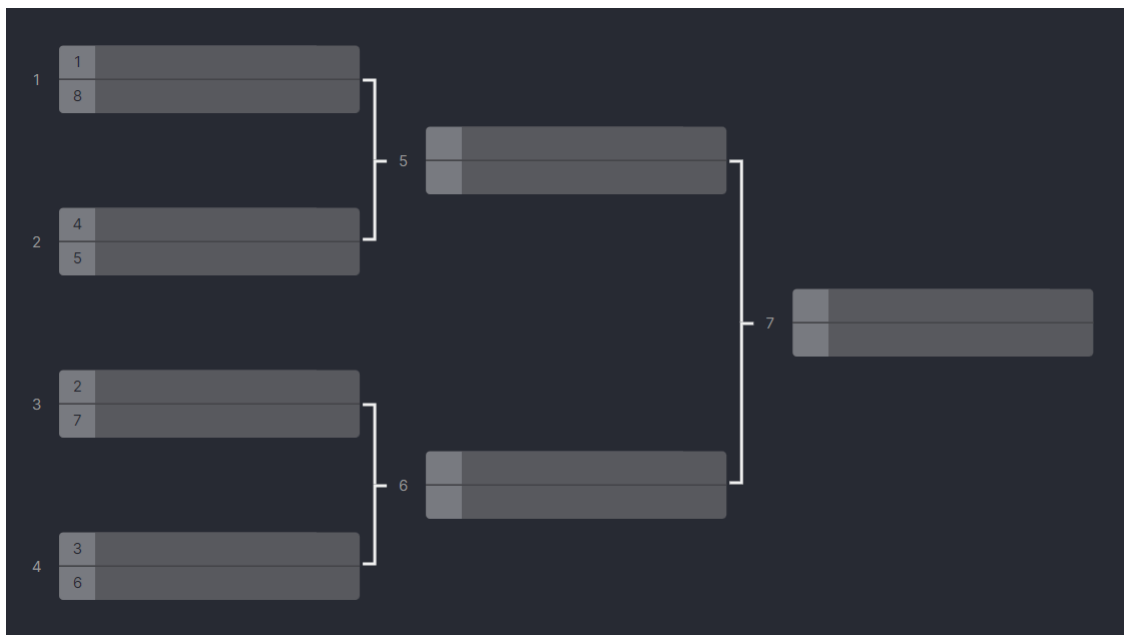


Abbildung 46: Turnierbaum[33]

## Phases

Ein Turnier ist in Phasen aufgeteilt, in der Abbildung oben ist jede Spalte von Matches eine Phase. Die Phasen eines Turniers werden mit der folgenden Methode eingefügt.

```

private void insertPhasesElimination(Tournament tournament, List<Competitor> competitors) {
    double numOfPhases = (Math.log(competitors.size())
        / Math.log(2));
    // numOfPlayers = 2^numOfPhases
    // solve for phases -> numOfPhases = log2(numOfCompetitors) = log(numOfCompetitors) / log(2)
    for (int i = 0; i < numOfPhases; i++) {
        phaseRepository.add(new Phase(i, groupName: -1, tournament));
    }
}

```

Abbildung 47: Einfügen der Phasen im Elimination Modus

Zuerst wird die Anzahl der Phasen anhand der Teilnehmeranzahl berechnet. Dies erfolgt mit dem Logarithmus zur Basis 2 von der Teilnehmeranzahl ( $\log_2(n)$ ). Da es in Java nicht möglich ist, die Basis eines Logarithmus frei zu wählen, wird stattdessen der Logarithmus der Teilnehmeranzahl dividiert durch den Logarithmus der Basis gerechnet, was zum gleichen Ergebnis kommt ( $\log(n)/\log(2)$ ). Anschließend werden die Phasen nummeriert in die Datenbank gespeichert, von 0 beginnend.

## Nodes

Als nächstes werden zu jeder Phase Nodes eingefügt, diese bilden zusammen mit den Phasen das Grundgerüst des Turniers.

```
private void insertNodesElimination(Tournament tournament) {
    List<Phase> phases = phaseRepository.getByTournamentGroup(tournament.getId(), groupNumber: -1);
    List<Node> previousNodes = null;

    for (int i = 0; i < phases.size(); i++) {
        Phase phase = phases.get(phases.size() - 1 - i);
        int numOfNodes = (int) Math.pow(2, i);
        for (int u = 0; u < numOfNodes; u++) {
            Node node = new Node(u, phase);
            if (previousNodes != null) {
                node.setNextNode(previousNodes.get(u / 2));
            }
            nodeRepository.add(node);
        }
        previousNodes = nodeRepository.getByPhaseId(phase.getId());
    }
}
```

Abbildung 48: Einfügen der Nodes im Elimination Modus

Es wird rückwärts durch die Phasen des Turniers iteriert, sodass, die letzte Phase am Ende nur einen Node hat, nämlich das Finale. Anschließend wird die Anzahl an Nodes für die momentane Phase berechnet, dies erfolgt mit der Rechnung 2 Hoch die Teilnehmeranzahl ( $2^n$ ). Anschließend wird für jeden Node der "nextNode" gesetzt, das ist jener Node, in den der Sieger des Matches aus dem momentanen Nodes aufsteigen würde, und diese dann in die Datenbank eingefügt.

## Matches

Zu jedem Node im Turnierbaum gehört ein Match, diese werden mit der folgenden Methode eingefügt.

```

private void insertMatchesElimination(Tournament tournament, List<Competitor> competitors) {
    List<Phase> phases = phaseRepository.getByTournamentGroup(tournament.getId(), groupName: -1);
    if (phases.size() < 1) {
        return;
    }
    competitors = getCompetitorsSeeded(tournament, competitors);
    List<Node> nodes = nodeRepository.getByPhaseId(phases.get(0).getId());
    for (int i = 0; i < competitors.size(); i++) {
        Node node = nodes.get(i / 2);
        if (node.getMatch() == null) {
            Match match = matchRepository.add(new Match());
            node.setMatch(match);
            nodeRepository.modify(node.getId(), node);
        }
        if (node.getMatch().getCompetitor1() == null) {
            node.getMatch().setCompetitor1(competitors.get(i));
        } else if (node.getMatch().getCompetitor2() == null) {
            node.getMatch().setCompetitor2(competitors.get(i));
        }
        matchRepository.modify(node.getMatch().getId(), node.getMatch());
    }
}

```

Abbildung 49: Einfügen der Matches im Elimination Modus

Diese Methode iteriert durch alle Nodes aus der ersten Phase durch, wobei durch "nodes.get(i / 2)" jede Node immer 2 Mal durchlaufen wird. Sollte eine Node noch kein Match haben, wird dieses hinzugefügt. Anschließend werden die Teilnehmer/innen, einer pro Durchlauf, also 2 pro Match, in die Matches eingefügt.

## Seeding

Um Turniere so spannend wie möglich zu gestalten, wird das sogenannte "Seeding" angewendet. Der Seed eines/einer Teilnehmers/in ist eine Einschätzung, wie gut dieser/diese Teilnehmer/in im Vergleich zu den anderen im selben Turnier ist. Somit ist der Seed 1 der/die beste Teilnehmer/in des Turniers, der Seed 2 der/die zweitbeste, usw. Mit Seeding will man erreichen, dass, angenommen der/die Teilnehmer/in mit dem niedrigeren Seed gewinnt immer gegen den/die mit dem höheren Seed, die Seeds 1 und 2, also die besten Teilnehmer/innen im Finale gegeneinander antreten. Würde man die Seeds einfach von oben nach unten in das Turnier einfügen, würde folgendes passieren.



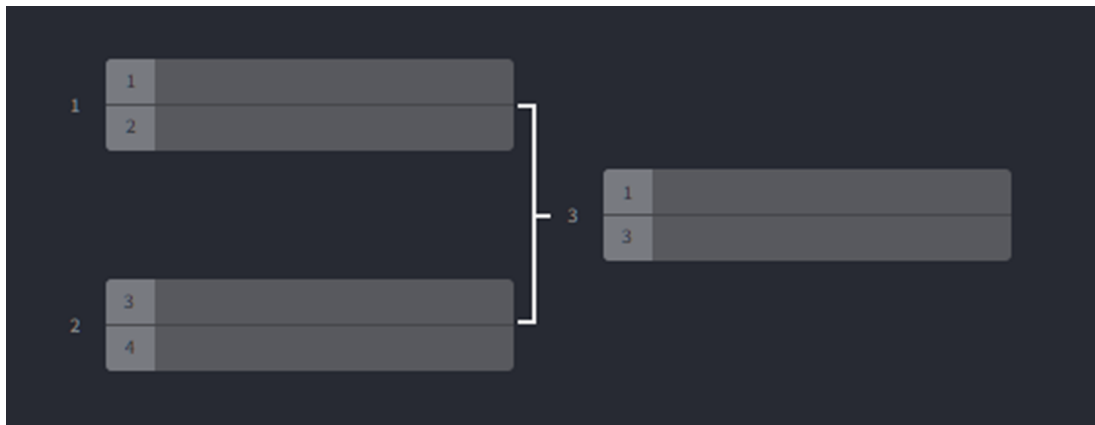


Abbildung 50: Turnierbaum falsch geseedet[33]

Wie man sieht, treten die ersten beiden Seeds bereits im Halbfinale gegeneinander an, was für ein möglichst spannendes Turnier nicht wirklich wünschenswert ist. Aus diesem Grund werden die Teilnehmer/innen folgendermaßen zusammengeführt.

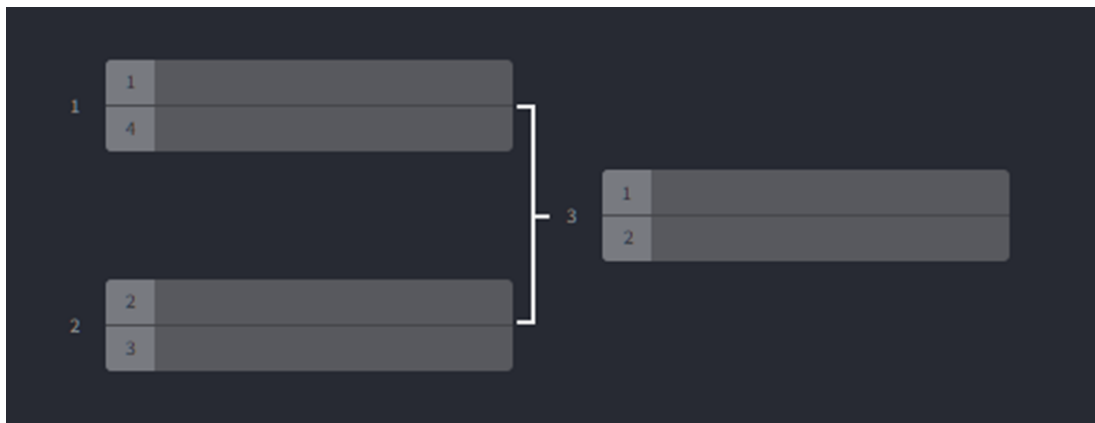


Abbildung 51: Turnierbaum richtig geseedet[33]

Der erste Seed tritt gegen den letzten an, der zweite gegen den vorletzten, usw. Auf diese Weise ist es am wahrscheinlichsten, dass das finale Match so spannend wie möglich wird. Es steckt jedoch noch ein bisschen mehr dahinter. Sollte man es so machen, dass im ersten Match von oben der erste Seed gegen den letzten antritt, im zweiten von oben der zweite gegen den vorletzten, usw., würde bei größeren Turnieren spätestens in der zweiten Phase das selbe Problem nochmal auftreten.

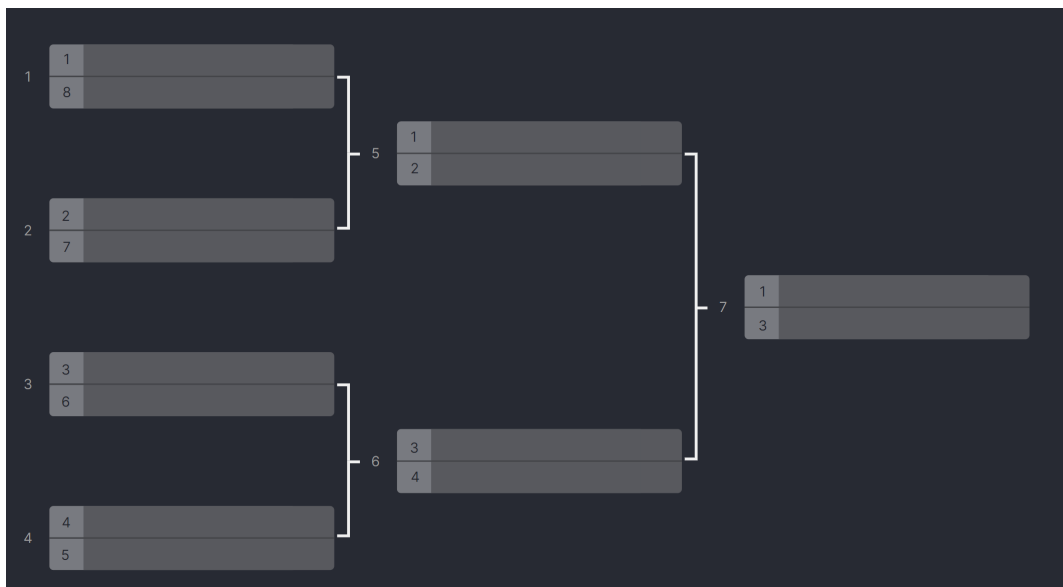


Abbildung 52: großer Turnierbaum falsch geseedet[33]

Um dieses Problem zu beheben, wird beim Anordnen Rekursion verwendet.

```
private List<Competitor> orderSeededCompetitors(List<Competitor> competitors) {
    if ((Math.log(competitors.size()) / Math.log(2)) % 1 != 0 && competitors.size() > 1) {
        return competitors;
    }
    return orderSeededCompetitors(competitors, level: 1);
}

private List<Competitor> orderSeededCompetitors(List<Competitor> competitors, int level) {
    level = level * 2;
    int numOfCompetitors = competitors.size();
    List<Competitor> res = new LinkedList<>();
    for (int i = 0; i < numOfCompetitors / level; i++) {
        for (int u = 0; u < level / 2; u++) {
            res.add(competitors.get(0));
            competitors.remove(i: 0);
        }
        for (int u = 0; u < level / 2; u++) {
            res.add(competitors.get(competitors.size() - 1));
            competitors.remove(i: competitors.size() - 1);
        }
    }
    if (level < numOfCompetitors) {
        res = orderSeededCompetitors(res, level);
    }
    return res;
}
```

Abbildung 53: Anordnen der Teilnehmer im Code

In der oberen Methode wird geprüft, ob die Anzahl der Teilnehmer/innen eine Hochzahl von 2 ist. Außerdem wird angenommen, dass die Teilnehmer/innen nach Seed aufsteigend

sortiert wurden. Nun wird die untere Methode ausgeführt, welche wie schon erwähnt Rekursion beinhaltet. Diese wird mit den folgenden Abbildungen erklärt.



Abbildung 54: Anordnen der Teilnehmer 1



Abbildung 55: Anordnen der Teilnehmer 2

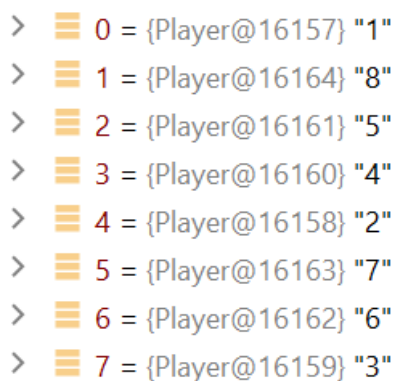


Abbildung 56: Anordnen der Teilnehmer 3

Im Grunde bewirkt dieser Code das gleiche, wie die Lösung von vorhin, nur mehrmals. Angenommen das Turnier hat 8 Teilnehmer. Der erste Durchlauf der Methode kümmert sich um die erste Phase, es werden die Seeds 1 und 8 zusammengeführt, 2 und 7, 3 und 6, und 4 und 5. Der zweite Durchlauf kümmert sich nun um die zweite Phase. In der ersten Phase wird wie erwähnt davon ausgegangen, dass die Seeds 1 bis 4 in

die zweite Phase aufsteigen, also sollten in der Phase die Seeds 1 und 4, und 2 und 3 zusammengeführt werden. Dies wird erreicht, indem in der ersten Phase die Seeds 1 und 8 (also in der zweiten Phase nur Seed 1) mit den Seeds 4 und 5 (also in der zweiten Phase nur Seed 4) zusammengeführt werden. Das ganze wird so lange wiederholt, bis für jede zukünftige Phase richtig angeordnet wurde. Im Turnierbaum sieht es dann so aus.

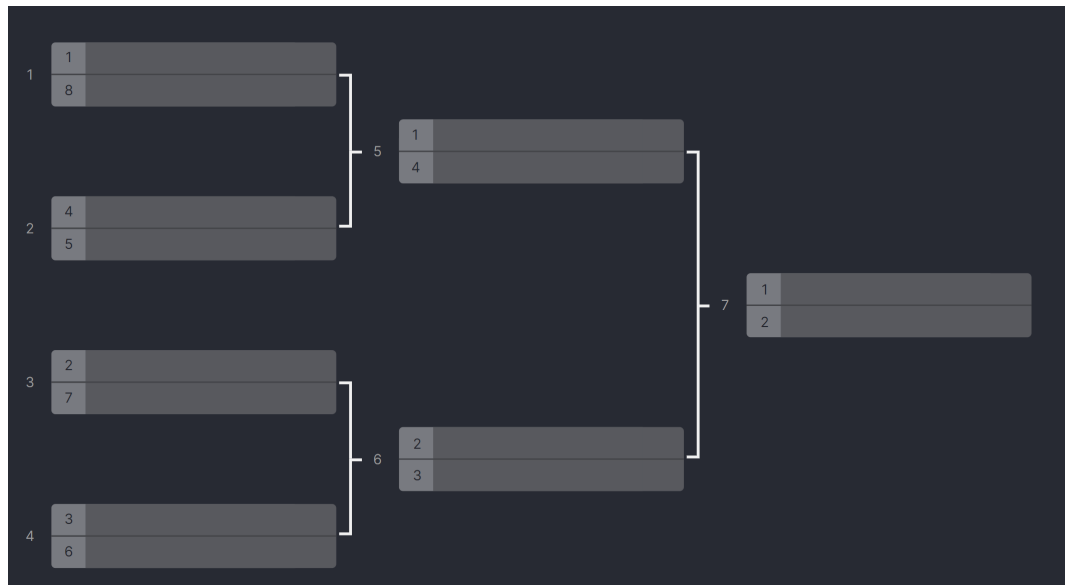


Abbildung 57: großer Turnierbaum geseedet[33]

Im Backend gibt es 2 Möglichkeiten, Seeds and Teilnehmer/innen zu verteilen. Die erste Möglichkeit ist es, die Teilnehmer nach der durchschnittlichen Platzierung aus vergangenen Turnieren zu sortieren, jedoch setzt das voraus, dass die Teilnehmer schon an Turnieren teilgenommen haben. Sollte dies nicht der Fall sein, ist die zweite Möglichkeit, dass die Teilnehmer manuell vom Turnierorganisator sortiert werden. Hier ist es allerdings von großer Bedeutung, einen vertrauenswürdigen Turnierorganisator haben, denn, wie man oben sieht, je besser man im Vergleich zu anderen Teilnehmern eingeschätzt wird, desto leichtere Gegner wird man bekommen.

### Buy-Rounds

Sollte die Anzahl der Teilnehmer/innen in einem Turnier keine Hochzahl von 2 sein, ist es nicht möglich, in der ersten Phase jedem/jeder Teilnehmer/in einen Gegner zu geben. Um dieses Problem zu lösen, steigen manche Teilnehmer automatisch in die zweite Phase auf, ohne ein Match gewonnen zu haben.

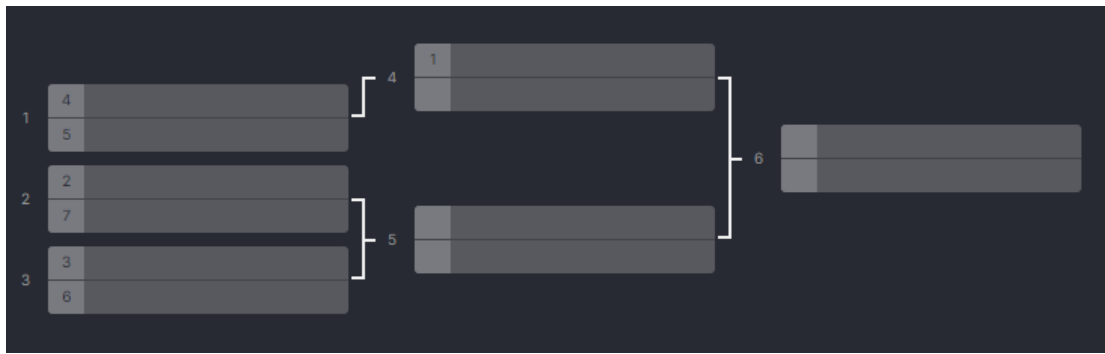


Abbildung 58: Turnierbaum mit Buy-Rounds[33]

Im Code wird das mit diesen Methoden erreicht.

```
private void setBuyRoundsElimination(Tournament tournament) {
    Phase phase = phaseRepository.getByTournamentGroup(tournament.getId(), groupNumber: -1).get(0);
    List<Node> nodes = nodeRepository.getByPhaseId(phase.getId());
    nodes.forEach(n -> {
        if (n.getMatch().getCompetitor1() == null || n.getMatch().getCompetitor2() == null) {
            Node nextNode = n.getNextNode();
            nextNode.setMatch(n.getMatch());
            nodeRepository.delete(n.getId());
            nodeRepository.modify(nextNode.getId(), nextNode);
        }
    });
}

private List<Competitor> getCompetitorsSeeded(Tournament tournament, List<Competitor> competitors) {
    // sort by seed
    sortBySeed(tournament, competitors);
    // add dummies
    int numOfPhases = phaseRepository.getByTournamentGroup(tournament.getId(), groupNumber: -1).size();
    int numOfCompetitors = competitors.size();
    for (int i = 0; i < Math.pow(2, numOfPhases) - numOfCompetitors; i++) {
        competitors.add(null);
    }

    // seed competitors
    competitors = orderSeededCompetitors(competitors);

    return competitors;
}
```

Abbildung 59: Buy-Rounds im Elimination Modus

Die untere Methode gibt alle Teilnehmer/innen des Turniers zurück, und zwar bereits nach Seed angeordnet. Besonders wichtig ist für Buy-Rounds der mittlere Teil der Methode. Angenommen ein Turnier hat 5 Teilnehmer/innen. Die nächste Hochzahl wäre in dem Fall 8, also befüllt diese Methode die Teilnehmerliste mit 3 "dummy" Teilnehmern, sodass die Teilnehmeranzahl eine Hochzahl von 2 ist. Dies ist notwendig, um die Teilnehmer/innen korrekt nach Seed anzuordnen.

Nun sind jedoch 3 "dummy" Teilnehmer in der Teilnehmerliste, mit denen man nichts anfangen kann. Um diese kümmert sich nun die obere Methode. Jeder/jede Teilneh-

mer/in, der einen "dummy" Teilnehmer als Gegner hat, steigt automatisch in die zweite Phase auf, und spielt dort das erste Match.

## Plazierungen

Die Platzierung eines/einer Teilnehmer/in in einem Turnier wird im Backend zu dem Zeitpunkt festgelegt, wenn der/die Teilnehmer/in ausgeschieden ist, oder das Turnier gewonnen hat. Berechnet wird die Platzierung durch die Anzahl der Teilnehmer in der nächsten Phase + 1, außer es ist das Finale, dann wird einfach entweder Platz 1 oder 2 vergeben. Sollte also ein/eine Teilnehmer/in in dem Top 8 aus dem Turnier ausscheiden, ist dieser/diese als 5ter platziert.

### 7.2.7 Round Robin

Im Round Robin Turniermodus tritt jeder/jede Teilnehmer/in gegen jeden Teilnehmer an, egal, wie oft man gewinnt oder verliert. Somit entsteht hier kein Turnierbaum, sondern eher ein Turnierraster aus Matches.



Abbildung 60: Turnierraster[33]

## Phases

Die Phasenanzahl im Round Robin Turniermodus ist etwas einfacher zu errechnen, als bei Elimination. Angenommen ein Turnier hat 4 Teilnehmer/innen, somit hat jeder/jede Teilnehmer/in insgesamt 3 Gegner. Da jeder/jede Teilnehmer/in in jeder Phase ein Match spielen kann, gibt es hier 3 Phasen, also die Teilnehmeranzahl - 1.

Bei einer ungeraden Teilnehmeranzahl ist es jedoch etwas anders, da in jeder Phase ein/eine Teilnehmer keinen Gegner hat. Daher wäre hier die Phasenanzahl gleich der Teilnehmeranzahl.



Abbildung 61: Turnierraster mit ungerader Teilnehmeranzahl[33]

Die Rechnung im Code sieht folgendermaßen aus.

```
double numOfPhases = competitors.size() - 1 + (competitors.size() % 2);
```

Abbildung 62: Einfügen der Phasen im Round Robin Modus

## Nodes und Matches

Da im Round Robin Turniermodus schon im Vorhinein bekannt ist, wer wann gegen wen antritt, werden hier Nodes und Matches gleichzeitig eingefügt. Dafür ist diese Methode verantwortlich.

```

public void insertNodesAndMatchesRoundRobin(Tournament tournament, List<Competitor> competitors,
                                             int groupName, int startingPhaseNumber) {
    List<Phase> phases = phaseRepository.getByTournamentGroup(tournament.getId(), groupName);
    int numOfNodes = competitors.size() / 2;
    for (int i = startingPhaseNumber; i < phases.size(); i++) {
        Phase phase = phases.get(i);
        List<Competitor> competitorsTmp = new ArrayList<>(competitors);
        if (competitorsTmp.size() % 2 != 0) {
            competitorsTmp.remove(i % competitorsTmp.size());
        }
        for (int u = 0; u < numOfNodes; u++) {
            Match match = new Match(competitorsTmp.remove(i % 0),
                                    competitorsTmp.remove(i % competitorsTmp.size()));
            if (startingPhaseNumber > 0) {
                nodeRepository.add(new Node( nodeNumber: -1, match, phase));
            } else {
                nodeRepository.add(new Node(u, match, phase));
            }
        }
    }
}

```

Abbildung 63: Einfügen der Nodes und Matches im Round Robin Modus

Ein/eine Teilnehmer/in muss in jeder Phase gegen verschiedene Teilnehmer/innen antreten. In der ersten Phase nimmt die Methode aus der Teilnehmerliste den/die erste/n Teilnehmer/in, und den/die unmittelbar nächste, und fügt sie zusammen, so lange, bis keine Teilnehmer/innen mehr in der Liste sind.

```

> 0 = {Player@16209} "1"
> 1 = {Player@16210} "2"
> 2 = {Player@16211} "3"
> 3 = {Player@16212} "4"

```

Abbildung 64: Round Robin Algorithmus 1

In der zweiten Phase wird wieder erste der/die Teilnehmer/in genommen, und diesmal der/die übernächste in der Liste.

```

> 0 = {Player@16209} "1"
> 1 = {Player@16210} "2"
> 2 = {Player@16211} "3"
> 3 = {Player@16212} "4"

```

Abbildung 65: Round Robin Algorithmus 2

Dann wird der/die erste und der überübernächste, zusammengeführt.

```

> 0 = {Player@16209} "1"
> 1 = {Player@16210} "2"
> 2 = {Player@16211} "3"
> 3 = {Player@16212} "4"

```

Abbildung 66: Round Robin Algorithmus 3



Das ganze wird so lange wiederholt, bis jeder/jede Teilnehmer/in gegen jeden/jede andere Teilnehmer/in angetreten ist.

### Tie breakers

Um zu vermeiden, dass am Ende eines Turniers 2 oder mehr Teilnehmer/innen gleich viele Matches gewonnen haben, und so auf dem gleichen Platz sind, wird bei den Plazierungen die Anzahl an Punkten ebenfalls in Betracht gezogen, jedoch besteht immernoch die Chance, dass sie ebenfalls die gleiche Anzahl an Punkten erzielt haben. Sollte dies der Fall sein, ist es notwendig, sogenannte "Tie Breakers" auszuführen. Tie Breakers sind Matches, die nach den Hauptmatches des Turnier noch extra ins Turnier hinzugefügt werden, um den Punktegleichstand zwischen den Teilnehmer/innen zu beheben. Diese zählen für die Platzierung nicht so viel wie ein normales Match, es kann also nicht sein, dass ein/eine Teilnehmer/in durch ein Tie Breaker Match noch höher Plaziert wird, als jemand der bei den normalen Matches nur knapp besser abgeschnitten hat als er/sie selbst.

### Plazierung

Die Platzierung im Round Robin Turniermodus wird festgesetzt, indem Teilnehmer zurest anhand der Differenz der Siege und Niederlagen sortiert werden, dann nach der Gesamtanzahl der Punkte, die von den/der Teilnehmer/in erzielt wurden, und dann nach Differenz der Siege und Niederlagen aus Tie Breaker Matches.

```
competitorDtos.sort((c1, c2) -> {  
    if (c1.getWins() > c2.getWins()) {  
        return -1;  
    } else if (c1.getWins() < c2.getWins()) {  
        return 1;  
    } else if (c1.getPoints() > c2.getPoints()) {  
        return -1;  
    } else if (c1.getPoints() < c2.getPoints()) {  
        return 1;  
    } else if (c1.getTieBreakerWins() > c2.getTieBreakerWins()) {  
        return -1;  
    } else if (c1.getTieBreakerWins() < c2.getTieBreakerWins()) {  
        return 1;  
    }  
    return 0;  
});
```

Abbildung 67: Platzierung im Round Robin Turniermodus

## 7.2.8 Combination

Der Combination Turniermodus ist eine Kombination aus Round Robin und Elimination. Zuerst werden die Teilnehmer/innen in Gruppen aufgeteilt, diese werden genau so durchgeführt wie im Round Robin Turniermodus, jeder/jede Teilnehmer/in tritt gegen jeden/jede andere/n Teilnehmer/in aus der gleichen Gruppe an. Anschließend steigen die besten Teilnehmer/innen aus der Gruppenphase in die KO-Phase auf, welche genau so durchgeführt wird, wie im Elimination modus, der Verlierer scheidet aus, der Gewinner steigt auf.

### Gruppenphase

In diesem Turniermodus wird größtenteils der Code für die anderen beiden Turniermodi wiederverwendet. Somit kann die Gruppenphase einfach gestartet, indem pro Gruppe einmal die "startTournament" Methode des Round Robin Turniermodus aufgerufen, welcher die momentane Gruppennummer übergeben wird. Die Anzahl der Gruppen kann hier frei gewählt werden.

```
public Tournament startGroupPhase(Tournament tournament, List<Competitor> competitors, int numOfGroups) {
    if (numOfGroups > competitors.size()) {
        return null;
    }
    eliminationExecution.sortBySeed(tournament, competitors);
    boolean x = false;
    for (int i = 0; i < numOfGroups; i++) {
        List<Competitor> competitorsInGroup = new LinkedList<>();
        for (int u = 0; u < competitors.size(); u++) {
            if (u % numOfGroups == 0) {
                x = !x;
            }
            if (x) {
                if ((u - i) % numOfGroups == 0) {
                    competitorsInGroup.add(competitors.get(u));
                }
            } else {
                if ((u - (numOfGroups - i - 1)) % numOfGroups == 0) {
                    competitorsInGroup.add(competitors.get(u));
                }
            }
        }
        roundRobinExecution.startTournament(tournament, competitorsInGroup, i);
    }
    return tournament;
}
```

Abbildung 68: Start der Gruppenphase im Combination Modus

Um zu vermeiden, dass in einer Gruppe bessere Teilnehmer/innen sind, als in einer anderen, werden Teilnehmer/innen anhand vom Seed gleichmäßig in die Gruppen verteilt. Wie das funktioniert, ist am folgenden Beispiel ersichtlich.

Teilnehmer/in	Gruppe
> 0 = {Player@16181} "1"	1
> 1 = {Player@16182} "2"	2
> 2 = {Player@16183} "3"	3
> 3 = {Player@16184} "4"	4
> 4 = {Player@16185} "5"	4
> 5 = {Player@16186} "6"	3
> 6 = {Player@16187} "7"	2
> 7 = {Player@16188} "8"	1
> 8 = {Player@16189} "9"	1
> 9 = {Player@16190} "10"	2
> 10 = {Player@16191} "11"	3
> 11 = {Player@16192} "12"	4
> 12 = {Player@16193} "13"	4
> 13 = {Player@16194} "14"	3
> 14 = {Player@16195} "15"	2
> 15 = {Player@16196} "16"	1

Abbildung 69: Verteilung in Gruppen

Hier sind in einem Turnier 16 Teilnehmer, welche auf 4 Gruppen aufgeteilt werden. Die ersten 4 werden nach der Reihe aufgeteilt, Seed 1 kommt in Gruppe 1, Seed 2 in Gruppe 2, usw. Die nächsten 4 werden in umgekehrter Reihenfolge aufgeteilt, das heißt, Seed 5 kommt in Gruppe 4, Seed 6 in Gruppe 3, usw. Das Ganze geht abwechselnd so weiter, bis alle Teilnehmer/innen verteilt wurden.

Sollten am Ende der Gruppenphase 2 oder mehr Teilnehmer/innen einer Gruppe die selbe Platzierung haben, können hier wieder die Tie Breaker Matches eingefügt werden. Wenn dies nicht der Fall ist, geht es weiter zu der KO-Phase.

```

public Tournament startKOPhase(Tournament tournament, int promotedPerGroup) {
    List<Competitor> promoted = new LinkedList<>();
    for (int i = 0; i < phaseRepository.getNumOfGroups(tournament.getId()); i++) {
        List<CompetitorDto> competitorDtos = roundRobinExecution.getCompetitorsSorted(tournament, i);
        for (int u = 0; u < promotedPerGroup; u++) {
            promoted.add(competitorRepository.getById(competitorDtos.get(u).getId()));
        }
    }
    return eliminationExecution.startTournament(tournament, promoted);
}

```

Abbildung 70: Start der KO-Phase im Combination Modus

Aus den Gruppen werden die Teilnehmer/innen, die am besten abgeschnitten haben, in einer Liste gespeichert, mit dieser wird dann die "startTournament" Methode des Elimination Turniermodus ausgeführt. Wie viele Teilnehmer/innen pro Gruppe aufsteigen, ist hier frei wählbar.

## Plazierungen

Die Teilnehmer/innen bekommen hier keine Platzierung ausgehend von der eigenen Gruppe, sondern vom gesamten Turnier. Für jene, die in bereits in der Gruppenphase ausgeschieden sind, wird die Platzierung so berechnet, wie im Round Robin Turniermodus, mit dem Unterschied, dass sie noch mit der Gruppenanzahl multipliziert wird. Sollte es also 16 Teilnehmer/innen in 4 Gruppen geben, teilen sich die mit der höchsten Platzierung pro Gruppe den vierten Platz. Die Teilnehmer/innen, die in die KO-Phase aufgestiegen sind, werden genauso wie im Elimination Turniermodus platziert.

## 7.3 KeyCloak

Damit das Backend auf den KeyCloak Server zugreifen kann, wird das "Quarkus OIDC" Library verwendet. Im "application.properties" File werden die URL des KeyCloak Realms sowie die Client Id festgelegt. Das Realm beinhaltet unter anderem die verschiedenen Rollen und User der Applikation, und außerdem den Client, dessen Client Id angegeben wird. Die Konfigurationen für KeyCloak, sowie die Rollen "admin" und "tournament-organizer", und ein "admin" User werden beim Start des Servers automatisch aus dem "realm.json" File importiert. Ein User hat eine Rolle, und kann auf alle für diese Rolle freigegebenen Endpoints zugreifen. Welche Rollen auf einen Endpoint zugreifen können wird mit der "@RolesAllowed" Annotation festgelegt.

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

Abbildung 71: RolesAllowed

Die Authentifizierung eines Users funktioniert mittels Bearer Token. Diesen bekommt man mit der folgenden Request.

POST ⌵ `http://localhost:8443/realms/leoturnier/protocol/openid-connect/token`

Params Authorization Headers (8) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

	KEY	VALUE
<input checked="" type="checkbox"/>	username	admin
<input checked="" type="checkbox"/>	password	admin
<input checked="" type="checkbox"/>	grant_type	password
<input checked="" type="checkbox"/>	client_id	leoturnier-client
	Key	Value

Abbildung 72: KeyCloak Token Request

Es ist ein Post Request mit der URL

`http://localhost:8443/realms/leoturnier/protocol/openid-connect/token`,

außerdem steht im Body der Request noch Username, Passwort sowie der Grant Type, der angibt, dass man Username und Passwort benötigt, und die Client Id. Den Token, den man durch diese Request bekommen hat, muss man nun in den Requests and die Endpoints des Quarkus Backends angeben, um darauf Zugriff zu erlangen.

Nun da die Implementierung der Applikation erklärt wurde, geht es mit dem Deployment der Applikation weiter.

# 8 Deployment

Hier wird beschrieben, wie die verschiedenen Softwarekomponenten, die nun fertig implementiert sind, einsatzbereit gemacht wurden.

Alle Komponenten der Applikation werden in jeweils einem Docker Container ausgeführt, um unabhängig vom Betriebssystem oder installierter Software (bis auf Docker) zu funktionieren. Damit die Container auch miteinander kommunizieren können, wurde im "docker-compose.yml" File ein Netzwerk namens "leoturnier" erstellt.

Das Frontend und Backend werden als Docker Image von einem Workflow auf Github Packages gepusht, welcher jedes Mal ausgeführt wird, wenn eine Änderung der Applikation auf Github gepusht wird. Die Docker Images werden mit "Dockerfiles" gebaut.

## 8.1 Backend

Das Dockerfile, das für das Docker Image des Backends benutzt wird, wurde von Quarkus automatisch generiert und sieht folgendermaßen aus.

```

FROM registry.access.redhat.com/ubi8/ubi-minimal:8.4

ARG JAVA_PACKAGE=java-11-openjdk-headless
ARG RUN_JAVA_VERSION=1.3.8
ENV LANG='en_US.UTF-8' LANGUAGE='en_US:en'
# Install java and the run-java script
# Also set up permissions for user `1001`
RUN microdnf install curl ca-certificates ${JAVA_PACKAGE} \
    && microdnf update \
    && microdnf clean all \
    && mkdir /deployments \
    && chown 1001 /deployments \
    && chmod "g+rwX" /deployments \
    && chown 1001:root /deployments \
    && curl https://repo1.maven.org/maven2/io/fabric8/run-java-sh/${RUN_JAVA_VERSION}/run-java-sh-${RUN_JAV,
    && chown 1001 /deployments/run-java.sh \
    && chmod 540 /deployments/run-java.sh \
    && echo "securerandom.source=file:/dev/urandom" >> /etc/alternatives/jre/conf/security/java.security

# Configure the JAVA_OPTIONS, you can add -XshowSettings:vm to also display the heap size.
ENV JAVA_OPTIONS="-Dquarkus.http.host=0.0.0.0 -Djava.util.logging.manager=org.jboss.logmanager.LogManager"
# We make four distinct layers so if there are application changes the library layers can be re-used
COPY --chown=1001 target/quarkus-app/lib/ /deployments/lib/
COPY --chown=1001 target/quarkus-app/*.jar /deployments/
COPY --chown=1001 target/quarkus-app/app/ /deployments/app/
COPY --chown=1001 target/quarkus-app/quarkus/ /deployments/quarkus/

EXPOSE 8080
USER 1001

ENTRYPOINT [ "/deployments/run-java.sh" ]

```

Abbildung 73: Dockerfile Backend

Im Grunde wird hier nur Java installiert und die für die Applikation wichtigen Files und Directories aus dem Backend in das Image kopiert.

## 8.2 Frontend

Das Dockerfile, das für das Docker Images des Frontends benutzt wird, entstand aus einer Vorlage eines bereits existierendes Dockerfile aus dem Internet [34]. Hier wird eigentlich nichts anderes gemacht, als Node und Nginx herunterladen um alle Libraries zu installieren, sowie alle wichtigen Files für die Angular App in das Image zu kopieren. Die einzige Änderung die gemacht werden musste, war die Einbindung des "nginx.conf" um das Routing in der Angular App zu ermöglichen.

```

### STAGE 1 ###
FROM node:14-alpine3.10 as build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . /app
# RUN ls

### STAGE 2 ###
FROM nginx:stable-alpine
COPY nginx/nginx.conf /etc/nginx/nginx.conf
COPY --from=build /app/dist/leo-turnier-frontend /usr/share/nginx/html
EXPOSE 4200:80
CMD ["nginx", "-g", "daemon off;"]

```

Abbildung 74: Dockerfile Frontend

## 8.3 Workflow

Im Workflow wird dann die Registry angegeben, in der die Images gepusht werden sollen, in dem Fall Github Packages, anschließend werden die Images gebaut und dann gepusht.

```

build_backend:
  name: Build backend
  runs-on: ubuntu-latest
  env:
    IMAGE_NAME: leo-turnier-backend
  defaults:
    run:
      working-directory: ./prototype/leo-turnier-backend
  steps:
    - name: Check out the repo
      uses: actions/checkout@v2
    - name: Package
      run: mvn package -DskipTests
    - name: Login to GitHub Packages
      uses: docker/login-action@v1
      with:
        registry: ghcr.io
        username: ${ github.actor }
        password: ${ secrets.GITHUB_TOKEN }
    - name: Build image
      run: docker build . -f src/main/docker/Dockerfile.jvm --tag $IMAGE_NAME
    - name: Push image
      run: |
        IMAGE_ID=ghcr.io/${ github.repository_owner }/$IMAGE_NAME
        IMAGE_ID=$(echo $IMAGE_ID | tr '[:upper:]' '[:lower:]')
        VERSION=latest
        echo IMAGE_ID=$IMAGE_ID
        echo VERSION=$VERSION
        docker tag $IMAGE_NAME $IMAGE_ID:$VERSION
        docker push $IMAGE_ID:$VERSION

```

Abbildung 75: Workflow Backend



```

build_frontend:
  name: Build frontend
  runs-on: ubuntu-latest
  env:
    IMAGE_NAME: leo-turnier-frontend
  defaults:
    run:
      working-directory: ./prototype/leo-turnier-frontend
  steps:
    - name: Check out the repo
      uses: actions/checkout@v2
    - name: Login to GitHub Packages
      uses: docker/login-action@v1
      with:
        registry: ghcr.io
        username: ${ github.actor }
        password: ${ secrets.GITHUB_TOKEN }
    - name: Build image
      run: docker build . --tag $IMAGE_NAME
    - name: Push image
      run: |
        IMAGE_ID=ghcr.io/${ github.repository_owner }/$IMAGE_NAME
        IMAGE_ID=$(echo $IMAGE_ID | tr '[A-Z]' '[a-z]')
        VERSION=latest
        echo IMAGE_ID=$IMAGE_ID
        echo VERSION=$VERSION
        docker tag $IMAGE_NAME $IMAGE_ID:$VERSION
        docker push $IMAGE_ID:$VERSION

```

Abbildung 76: Workflow Frontend

## 8.4 Docker-Compose

Ausgeführt werden die Images dann in einem Docker Container, der im dem "docker-compose.yml" File gestartet wird.

### Backend

```

backend:
  container_name: leoturnier_backend
  image: ghcr.io/htl-leonding-project/leo-turnier-backend:latest
  restart: unless-stopped
  ports:
    - "8080:8080"
  depends_on:
    - auth
  networks:
    - leoturnier

```

Abbildung 77: Docker Compose Backend

Hier wird das Docker Image, das im Vorhinein vom Workflow gepusht wurde, heruntergeladen und mit dem Port 8080 gestartet.

Zuvor wird noch die PostgreSQL Datenbank, in der die Turnierdaten gespeichert werden, gestartet.

```
db:
  container_name: leoturnier_db
  image: postgres:latest
  restart: unless-stopped
  environment:
    POSTGRES_USER: app
    POSTGRES_PASSWORD: app
    POSTGRES_DB: db
  ports:
    - "5432:5432"
  volumes:
    - ./db-postgres/import:/import
  networks:
    - leoturnier
```

Abbildung 78: Docker Compose Datenbank

## Frontend

```
frontend:
  container_name: leoturnier_frontend
  image: ghcr.io/htl-leonding-project/leo-turnier-frontend:latest
  restart: unless-stopped
  ports:
    - "4200:80"
  depends_on:
    - backend
  networks:
    - leoturnier
```

Abbildung 79: Docker Compose Frontend

## KeyCloak

Der KeyCloak Server sowie die Datenbank, in der die Userdaten verwaltet werden, werden ebenfalls im "docker-compose.yml" File gestartet.

```
auth:
  container_name: leoturnier_auth
  image: quay.io/keycloak/keycloak:latest
  restart: unless-stopped
  command:
    - start --import-realm --http-port 8443
  volumes:
    - ./imports:/opt/keycloak/data/import
  environment:
    DB_VENDOR: POSTGRES
    DB_ADDR: auth_db
    DB_DATABASE: auth_db
    DB_USER: app
    DB_SCHEMA: public
    DB_PASSWORD: app
    KEYCLOAK_ADMIN: admin
    KEYCLOAK_ADMIN_PASSWORD: admin
    KC_HTTP_ENABLED: 'true'
    KC_HOSTNAME_STRICT: 'false'
    KC_HOSTNAME_STRICT_HTTPS: 'false'
  ports:
    - "8443:8443"
  networks:
    - leoturnier
  depends_on:
    - auth_db
```

Abbildung 80: Docker Compose KeyCloak

Es werden die Konfigurationen importiert, der Port festgelegt, und User und Passwort für die verwendete Datenbank angegeben, welche zuvor ebenfalls im "docker-compose.yml" File gestartet wurde.

```
auth_db:
  container_name: leoturnier_auth_db
  image: postgres:latest
  restart: unless-stopped
  environment:
    POSTGRES_DB: auth_db
    POSTGRES_USER: app
    POSTGRES_PASSWORD: app
  networks:
    - leoturnier
```

Abbildung 81: Docker Compose Userdatenbank

## 9 Zusammenfassung

Während der Schulzeit arbeitet man zwar an vielen Projekten, diese erfordern jedoch im Vergleich zur Diplomarbeit weit nicht so viel Organisationsaufwand und Kommunikation, da die Diplomanden für längere Zeit an dem selben Projekt arbeiten. Dabei war es vor allem wichtig, gerade auf Github immer die aktuellste Version zu haben und Erweiterungen mit informativen Commit-Messages zu beschreiben.

Von den verwendeten Technologie wurde schon mit Einigen während der Schulzeit gearbeitet, wie zum Beispiel Angular und Quarkus, in diese wurde sich im Zuge der Diplomarbeit noch deutlich vertieft. Durch die getrennte Arbeitsaufteilung von Backend und Frontend, war die Technologien Postman und MockLab sehr wichtig um keine Unterbrechungen in der Entwicklung zu verursachen. Als besonders große Herausforderung stellte sich die Arbeit mit Keycloak heraus, beziehungsweise die generelle Arbeit mit Authentifizierung und Autorisierung in einer größeren Applikation.

Die Diplomanden haben im Verlauf der Diplomarbeit viel Erfahrung im Bereich Turniere gemacht, wie sie aufgebaut sind, wie sie so spannend wie möglich gestaltet werden, usw.

Eine besonders schwere Aufgabe war es, anfangs die Ausmaße unseres Projektes einzuschätzen und im Voraus dafür zu planen. Dabei wurden vor allem das Datenmodell im Projektverlauf mehrmals überarbeitet. Darum war es wichtig das der Code so geschrieben wurde, dass er auch nach längerer Zeit noch verständlich ist.

Mögliche Erweiterungen des Projekts könnten sein:

- Live-Ticker der mehr Einsicht in das Spielgeschehen gibt
- Detailreicher Statistiken über Spiele oder einzelne Teilnehmer
- Manuelle Anordnung der Seeds
- Ein Chat damit sich Zuschauer über das Spielgeschehen unterhalten können



# 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://reSTEeasy.dev/>
- [5] Redaktion ComputerWeekly.de, „Hibernate,” 2020, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.itwissen.info/Hibernate-hibernate.html>
- [6] , „object relational mapping (ORM),” 2020, letzter Zugriff am 06.09.2022. Online verfügbar: <https://www.itwissen.info/object-relational-mapping-ORM.html>
- [7] —, „Maven,” letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.computerweekly.com/de/definition/Maven>
- [8] Patrick Woods, „Was ist PostgreSQL?” 2021, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.plusserver.com/blog/was-ist-postgresql>
- [9] , „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/>
- [10] Tobias Surmann, „Keycloak – Ein Überblick,” letzter Zugriff am 21.08.2022. Online verfügbar: <https://www.smf.de/keycloak-ein-ueberblick/>
- [11] , „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&gclid=aw.ds](https://www.redhat.com/de/topics/containers/what-is-docker?sc_cid=7013a000002wLw0AAE&gclid=Cj0KCQjwjIKYBhC6ARIsAGEds-KwmuTopp7YzOB18IVrXiFvqwtPWlxMCQlrUTA9mDJokqoQZRqMcNcaAv5LEALw_wcB&gclid=aw.ds)
- [12] 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>
- [13] J. Tillmann, „Was ist eigentlich Typescript,” 2017, letzter Zugriff am 22.08.2022. Online verfügbar: <https://t3n.de/news/eigentlich-typescript-859869/>
- [14] , „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/>

- [15] —, „Was ist Git?“ letzter Zugriff am 18.08.2022. Online verfügbar: <https://www.atlassian.com/de/git/tutorials/what-is-git>
- [16] —, „GIT FACHBEGRIFFE EINFACH ERKLÄRT,“ letzter Zugriff am 18.08.2022. Online verfügbar: <https://www.arocom.de/fachbegriffe/webentwicklung/git>
- [17] 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>
- [18] , „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/>
- [19] 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.>
- [20] 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/>
- [21] , „Was ist IntelliJ IDEA?“ letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.jetbrains.com/de-de/idea/features/>
- [22] —, „Was ist eine IDE?“ 2019, letzter Zugriff am 20.08.2022. Online verfügbar: <https://www.redhat.com/de/topics/middleware/what-is-ide>
- [23] —, „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->
- [24] 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/>
- [25] , „Was ist Postman?“ letzter Zugriff am 22.08.2022. Online verfügbar: <https://www.testautomatisierung.org/lexikon/postman-api/>
- [26] —, „Was ist Mocklab,“ letzter Zugriff am 22.08.2022. Online verfügbar: <https://get.mocklab.io/%7D>
- [27] B. Landmesser, „App-Entwicklung ganz einfach: eine Einführung in Angular 2,“ 2022, letzter Zugriff am 30.08.2022. Online verfügbar: <https://www.cocomore.de/blog/app-entwicklung-ganz-einfach-eine-einfuehrung-angular-2>
- [28] awiseman, „What are the "spec.ts" files generated by Angular CLI for?“ 2022, letzter Zugriff am 30.08.2022. Online verfügbar: <https://stackoverflow.com/questions/37502809/what-are-the-spec-ts-files-generated-by-angular-cli-for>
- [29] , „Add services,“ 2022, letzter Zugriff am 30.08.2022. Online verfügbar: <https://angular.io/tutorial/toh-pt4#add-services>
- [30] —, „Communicating with backend services using HTTP,“ 2022, letzter Zugriff am 30.08.2022. Online verfügbar: <https://angular.io/guide/http>

- [31] K. Gorman, „Hauptunterschiede zwischen NgOnInit und Konstruktor-Methoden in Angular,” 2022, letzter Zugriff am 31.08.2022. Online verfügbar: <https://chudovo.de/hauptunterschiede-zwischen-ngoninit-und-konstruktor-methoden-in-angular/>
- [32] O. Happe, „NgTournamentTree,” 2019, letzter Zugriff am 31.08.2022. Online verfügbar: <https://www.npmjs.com/package/ng-tournament-tree>
- [33] „Tournament Bracket Generator,” letzter Zugriff am 25.08.2022. Online verfügbar: [https://challonge.com/tournament/bracket\\_generator](https://challonge.com/tournament/bracket_generator)
- [34] , „Dockerize Angular Application,” 2022, letzter Zugriff am 31.08.2022. Online verfügbar: <https://medium.com/codex/dockerize-angular-application-69e7503d1816>



# Abbildungsverzeichnis

1	System Architecture . . . . .	5
2	Postman Request . . . . .	25
3	Homepage . . . . .	27
4	Tree-View . . . . .	30
5	Table-View . . . . .	31
6	Login-Page . . . . .	32
7	Player-Overview-Page . . . . .	33
8	Player-Details-Page . . . . .	33
9	Team-Overview-Page . . . . .	34
10	Team-Players-PopUp . . . . .	34
11	Team-Details-Page . . . . .	35
12	Add-Player-PopUp (Team) . . . . .	35
13	Tournament-Overview-Page . . . . .	36
14	Tournament-Details-Page . . . . .	37
15	Tournament-Details-Page mit Add-Player Funktion . . . . .	37
16	Add-Player-PopUp (Tournament) . . . . .	38
17	Match-Overview-Page . . . . .	39
18	Match-Submission-Page . . . . .	40
19	Angular Ordnerstruktur . . . . .	42
20	package.json . . . . .	43
21	App Ordner . . . . .	44
22	routes in app-routing.module.ts . . . . .	45
23	@NgModule in app.module.ts . . . . .	45
24	player.model.ts . . . . .	46
25	tournament.service.ts . . . . .	47
26	Beispiel Request: startTournament . . . . .	47
27	app.init.ts . . . . .	47
28	Keycloak-Frontend Client . . . . .	48
29	Beispiel Router-Call aus player-overview.component.ts . . . . .	49
30	Beispiel Router-Call aus player-overview.component.ts . . . . .	49
31	Beispiel ActivatedRouter-Call aus player.component.ts . . . . .	49
32	hinzufügen ausstehender Matches in showTree(Home-Component) . . . .	50
33	Turnierbaum . . . . .	51
34	my-tree-configurations.scss . . . . .	51
35	style.scss . . . . .	51
36	Class Diagram . . . . .	52
37	Quarkus Ordnerstruktur . . . . .	53
38	pom.xml . . . . .	54
39	application.properties . . . . .	55
40	Entitätsklasse . . . . .	56
41	Repositoryklasse . . . . .	58
42	add Methode aus TournamentRepository . . . . .	59

43	Serviceklasse . . . . .	60
44	get Methode aus TournamentService . . . . .	61
45	ßstartTournament" Methode der Execution Klasse . . . . .	62
46	Turnierbaum[33] . . . . .	63
47	Einfügen der Phasen im Elimination Modus . . . . .	63
48	Einfügen der Nodes im Elimination Modus . . . . .	64
49	Einfügen der Matches im Elimination Modus . . . . .	65
50	Turnierbaum falsch geseedet[33] . . . . .	66
51	Turnierbaum richtig geseedet[33] . . . . .	66
52	großer Turnierbaum falsch geseedet[33] . . . . .	67
53	Anordnen der Teilnehmer im Code . . . . .	67
54	Anordnen der Teilnehmer 1 . . . . .	68
55	Anordnen der Teilnehmer 2 . . . . .	68
56	Anordnen der Teilnehmer 3 . . . . .	68
57	großer Turnierbaum geseedet[33] . . . . .	69
58	Turnierbaum mit Buy-Rounds[33] . . . . .	70
59	Buy-Rounds im Elimination Modus . . . . .	70
60	Turnierraster[33] . . . . .	71
61	Turnierraster mit ungerader Teilnehmeranzahl[33] . . . . .	72
62	Einfügen der Phasen im Round Robin Modus . . . . .	72
63	Einfügen der Nodes und Matches im Round Robin Modus . . . . .	73
64	Round Robin Algorithmus 1 . . . . .	73
65	Round Robin Algorithmus 2 . . . . .	73
66	Round Robin Algorithmus 3 . . . . .	73
67	Plazierung im Round Robin Turniermodus . . . . .	74
68	Start der Gruppenphase im Combination Modus . . . . .	75
69	Verteilung in Gruppen . . . . .	76
70	Start der KO-Phase im Combination Modus . . . . .	76
71	RolesAllowed . . . . .	77
72	KeyCloak Token Request . . . . .	78
73	Dockerfile Backend . . . . .	80
74	Dockerfile Frontend . . . . .	81
75	Workflow Backend . . . . .	81
76	Workflow Frontend . . . . .	82
77	Docker Compose Backend . . . . .	82
78	Docker Compose Datenbank . . . . .	83
79	Docker Compose Frontend . . . . .	83
80	Docker Compose KeyCloak . . . . .	83
81	Docker Compose Userdatenbank . . . . .	84

# Tabellenverzeichnis

# Quellcodeverzeichnis

# Anhang