INF20B Gruppe 5: 2447899, 1360712 , 9500721

Sports-WG

Dokumentation des Webservices



Bundesliga-Ergebnisservice mit integrierter Wettfunktion

Inhalt

[1 Projektidee und -ablauf 2](#_Toc98797735)

[1.1 Grundidee des Webservice „Bundesligaticker“ 2](#_Toc98797736)

[1.2 Projektplan 2](#_Toc98797737)

[2 BPMN-Modell 2](#_Toc98797738)

[2.1 Bundesliga – Spiele 2](#_Toc98797739)

[2.2 User 3](#_Toc98797740)

[2.3 Wetten 4](#_Toc98797741)

[3 Implementierung 5](#_Toc98797742)

[3.1 Frontend 5](#_Toc98797743)

[3.2 Backend 5](#_Toc98797744)

[3.2.1 Endpunkt 2: Ein Match anschauen 6](#_Toc98797745)

[3.2.2 Endpunkt 8: Registrieren 6](#_Toc98797746)

[3.2.3 Endpunkt 14: Wetten eintragen 7](#_Toc98797747)

[3.3 Datenbank 8](#_Toc98797748)

[4 Performancetest in SoapUI 8](#_Toc98797749)

# 1 Projektidee und -ablauf

## 1.1 Grundidee des Webservice „Bundesligaticker“

Als Prozess, den der geschriebene Webservice erweitern soll, wurde ein Sportwettprozess für Spiele der ersten Fußball-Bundesliga zu Grunde gelegt. Dazu wurde ein Web-Frontend zum Anzeigen des selbst entwickelten Webservices entwickelt, welcher es sowohl erlaubt, die Ergebnisse und diverse Bundesliga-Statistika seit 2015 anzeigen zu lassen, als auch möglich macht, ein Benutzerkonto zu erstellen und mit diesem Wetten auf anstehende Spiele abzuschließen.

Da je nach Spieltag entweder am Wochenende oder auch mal unter der Woche Spiele in der Bundesliga stattfinden, sollten sich die Daten für Tore, Torschützen, Wettquoten und Spiele in der Datenbank täglich aktualisieren.

## 1.2 Projektplan

Das Projekt begann am 08. Februar mit einem ersten Termin, als sich nach sämtlichen Ideen für verschiedene Webservices schließlich die „Bundesliga-Statistika“ mit Wettmöglichkeit durchsetzte. Wir entschieden uns die Tage darauf dann dafür, eine REST-API mithilfe von NodeJS und Express im Backend sowie Angular im Frontend zu erstellen. Eine Datenbank mit MySQL war dabei auch von Nöten, um die „historischen“ Daten ab 2015 im Petto zu haben. Die darauffolgenden Wochen arbeiteten Tim Schweitzer und Felix Schieferdecker hauptsächlich am Backend, dem Erstellen der Endpunkte sowie der Datenbank und Tillmann Lorenz hauptsächlich an der BPMN-Modellierung sowie am zugehörigen Frontend, also einer Web-Oberfläche für die besagten Endpunkte.

Anfang bis Mitte März wurde dann die übliche Zeit genutzt, um die Dokumentation anzufertigen und die Implementierung zu beenden.

# 2 BPMN-Modell

## 2.1 Bundesliga – Spiele

Diese Abbildung zeigt das BPMN-Modell, welches eine Abfrage eines Spieles mithilfe des OpenligaDB-Webservice darstellt. Der User wählt ein Spiel in der UI aus. Endpunkt 2 bekommt daraufhin die zugehörige Spiel-ID des OpenligaDB-Webservice[[1]](#footnote-1) und es wird ein SQL-SELECT-Statement für die Datenbankverbindung bereitgemacht. Über die Spiel-ID werden dann sämtliche Informationen über das gewünschte Spiel bereitgestellt. Wenn die Spiel-ID nicht existiert, wird entsprechend ein HTTP Status zurückgegeben. Wenn wiederum die ID nicht aus einer Zahlenfolge entspricht (und dementsprechend keine richtige ID ist) wird eine Meldung zurückgegeben, dass die Spiel-ID nicht in einem korrekten Format ist.

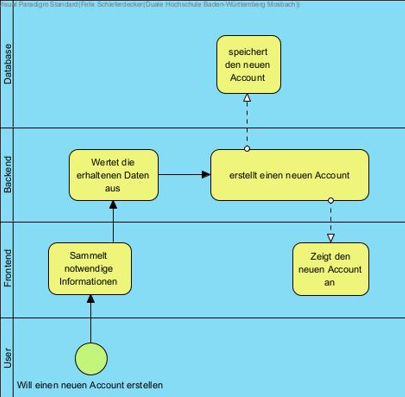


Abbildung 1

## 2.2 User

Die Abbildung auf der nächsten Seite zeigt das BPMN-Modell, welches den Registrierungsvorgang im Front- bzw. Backend sowie auf Datenbankebene zeigt. Zunächst will ein User einen neuen Account erstellen. Dazu muss er im Frontend eine E-Mail-Adresse sowie ein Passwort hinterlegen. Diese Daten werden dann über den Endpunkt 8 verarbeitet. Wenn die Daten valide sind, also die gleiche E-Mail nicht bereits existiert, dann wird ein neuer Account erstellt. In diesem Schritt werden diese Daten über ein SQL-Statement innerhalb des Endpunkts über ein „INSERT INTO“ in die entsprechende user-Tabelle in der Datenbank gespeichert. Sollte allerdings bereits ein User mit der gleichen E-Mail existieren, wird ein 204 HTTP Status mit der Nachricht „Bereits ein User mit dieser E-Mail“ zurückgegeben. Zuletzt wird der Kunde über den neu erstellten Account angemeldet und kann dessen Funktionen nutzen.

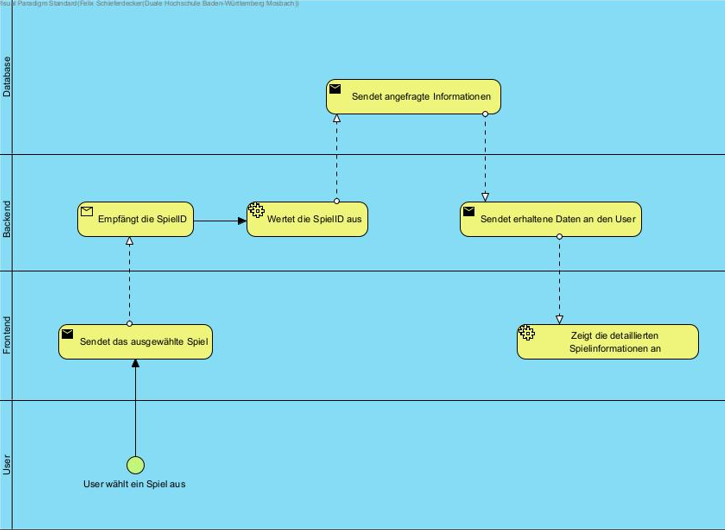


Abbildung 2

## 2.3 Wetten

Die Abbildung auf der nächsten Seite zeigt das BPMN-Modell, das den Wettvorgang eines Users mit dem Wettquotenwebservice darstellt. Wenn ein User eine Wette zu einem Spiel platzieren will, so werden ihm zunächst die zugehörigen Wettquoten angezeigt. Dies geschieht über den Wettquotenwebservice[[2]](#footnote-2) , welcher alle Wettquoten der bevorstehenden Spiele des nächsten Spieltags der Bundesliga zurückgibt. Da diese Wettquoten natürlich je nach Ansetzung und Spiel Woche für Woche variieren, werden diese über einen Cronjob alle 24 Stunden aktualisiert.

Entscheidet sich ein User nun für ein Spiel, für dass es sich zu Wetten lohnt, kann er einen bestimmten Betrag an Coins, welche er vorher eingezahlt hatte, in die UI eingeben. Über den 14. Endpoint werden die Daten ans Backend übergeben. Dieses überprüft zunächst den User und seinen Kontostand und ob er überhaupt so viel Geld zur Verfügung hat. Dann wird eine neue Wette in die Datenbank eingeben und abgespeichert. Die neuen Informationen werden dem Nutzer anschließend angezeigt.

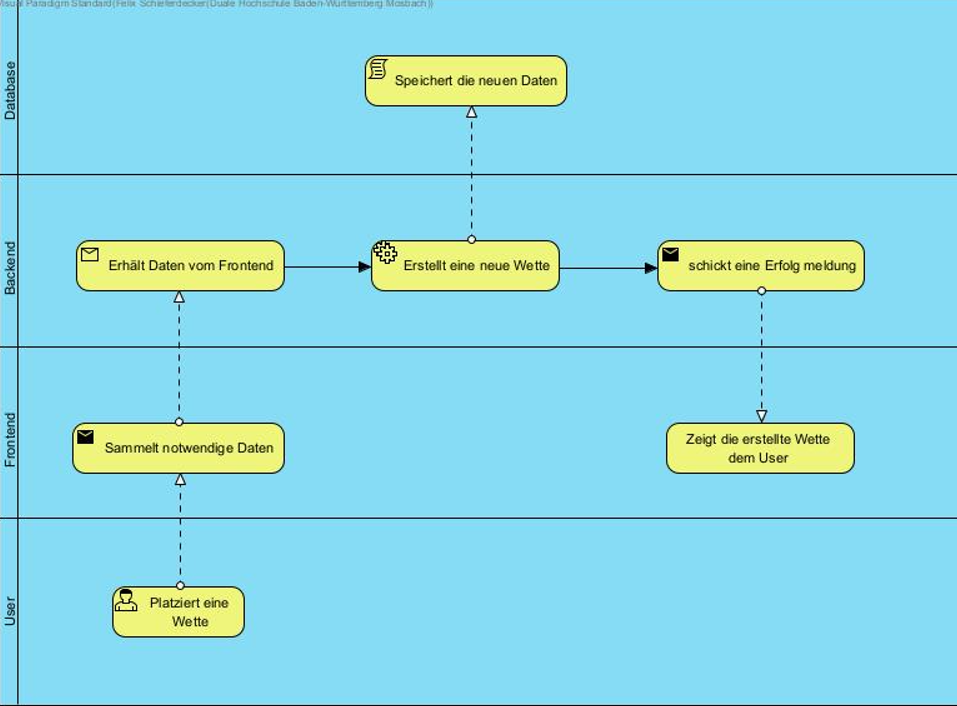


Abbildung 3

# 3 Implementierung

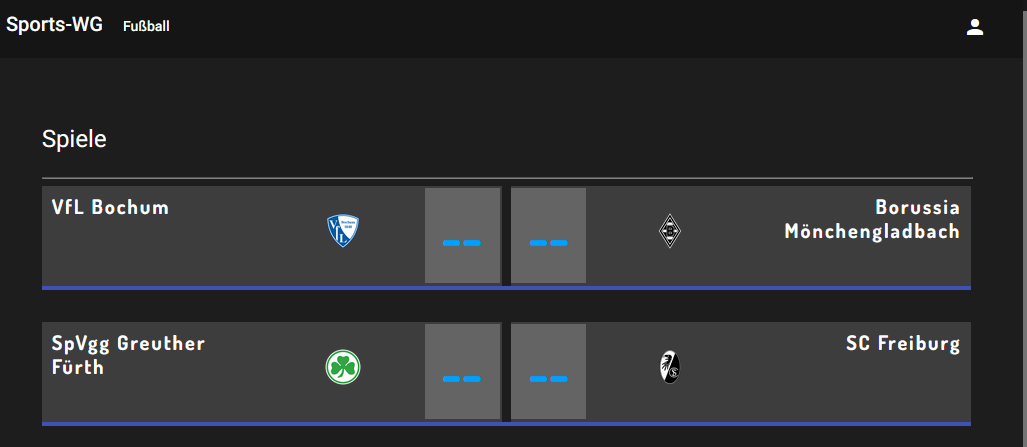
## 3.1 Frontend

Der Aufbau des Frontends bzw. das Design und die genutzten Techniken hängt wie beim Backend sehr stark von der Anforderung ab die man an die Webseite hat. An sich würde für das reine Bundesligaspiel eine statische Webseite reichen da wir statisch diese Webseiten laden und dann somit das einfache Aufrufen von Webseiten haben. Was ein wenig problematisch wäre bzw. nicht so benutzerfreundlich wäre das die Seiten bei jedem Aufruf neu geladen werden müssen. Dem widerspricht jedoch das wir ein Backend haben was nicht HTML Seiten ausgibt sondern JSON Dateien. Dadurch kann man schlecht darum ein Programm bauen das die Seiten nimmt und statisch einlädt wie PHP. Des Weiteren kann man in Abhängigkeit auf was man klickt verschiedene Dinge sehen auf der selben Webseite und durch das Verfahren des Logins gibt es ein Status der über der gesamten Webseite konsistent sein muss. Dieser Login Status muss zudem durchgehend konsistent sein. Wir wollen zudem die Anfragen an das Backend minimieren. Deswegen wird eine Umgebung genutzt die den Status der Webseite speichert. Sogenanntes Statefull Programming.

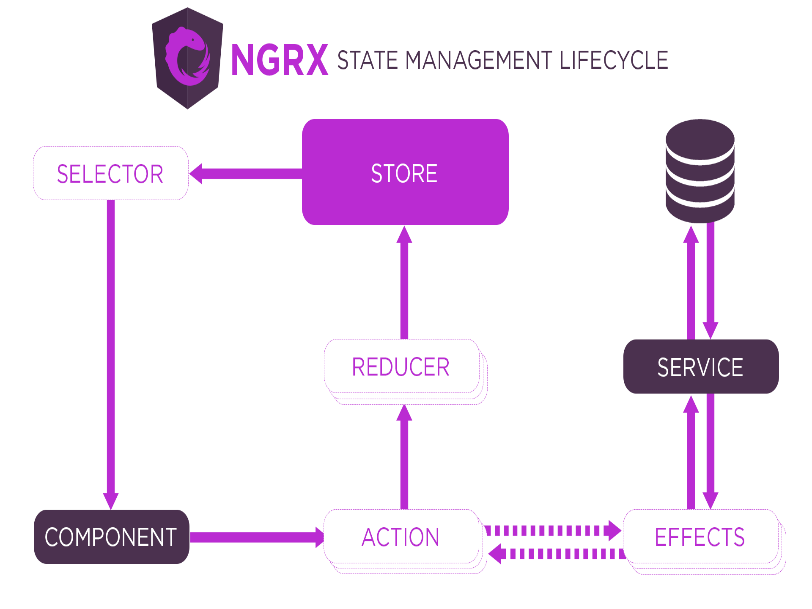
Dabei wird beim Statefull Programming darauf geachtet, dass die Web App ein State hat also ein Status wie sie gerade ist. Dieser kann durch Klicks oder durch externe Events wie das ändern eines Eintrags im Backend geändert werden. Für diese Art der Programmierung fallen grob 3 Programmiersprachen/Frameworks in Betracht: Native JS, Angular und React.

Native JS hat das Problem das es sehr kompliziert ist und zeitaufwendig und somit für den limitierten Rahmen den wir hatten unpassend ist. React hätte von dem Programmieren neu gelernt werden müssen und zudem besitzt React kein ausreichendes Subscribe Modell.

Angular ist für diese Möglichkeit perfekt geeignet. Es ist ein JavaScript bzw. TypeScript Framework. Dieses hat den Vorteil das alle Teile einer Webseite in Komponenten untergliedert werden und auch zudem eine gute Skalierbarkeit hat, also gut wäre für große Firmen. Was für uns den Charme ausgemacht hat ist das Angular durch das integrierte HttpClientModule die Möglichkeit bietet das automatisch zu Backend Services ein Abodenment aufgebaut wird und regelmäßig überprüft wird ob der Service bereits vorhanden ist.

Den Frontend Service kann man aufrufen über *<ip\_adresse des Docker Containers>*:4200. Dabei wird einem am Anfang alle Spiele der Zukunft angezeigt. Hierbei wird vom Backend der Endpuinkt /home aufgerufen und dann werden alle entsprechenden Bundesligaspiele geladen mit den dazugehörigen Ergebnissen und anhand des blauen Balkens kann man sehen wie viel Zeit vergangen ist.

Hierbei ist interessant zu merken das der Login Balken, die Hauptwebseite und die einzelnen Matches unterschiedliche Komponenten sind die ineinander verschachtelt sind. Dadurch kann erreicht werden das nur Teile der Webseite neu geladen werden müssen. Dies beschleunigt die Zeit der Angularanwendung.

Bei nicht erreichen des Backends oder bei Fehlerausgaben wird einfach nichts angezeigt. Allgemein gilt das bei Fehlschlägen sich die entsprechende UI nicht ändert, sondern einfach alles so bleibt wie bisher. Ein weiteres Feature ist das Login. Wenn man sich einloggt kommt eine weitere Komponente zum Einsatz, die hilft den Zustand der App über alle Komponenten hinweg gleich zu halten. Das hat den Vorteil das einerseits nicht im lokalen Storage abgespeichert werden muss aber auch die Komponenten mitbekommen, wenn z.B. abgemeldet wird und dadurch die Wetten nicht mehr angezeigt werden müssen. Das alles passiert ohne das neu Laden einer Seite. Dieses Modul das genutzt wird heißt Ngrx Store. Dabei wird hier kurz auf die Funktionsweise des Stores eingegangen zum besseren Verständnis.

Wie man rechts in der Abbildung sehen kann wird hier ein Store genutzt. Dieser hält den Zustand der gesamten App in sich bzw. den Zustand der wichtig ist. In unserem Fall sind es die Login Daten und Accountinformationen und eventuelle Error Codes. Dabei kann jede Komponente Actions ausführen die dann Auswirkungen haben auf den Store. Dabei kann z.B. bei dem klicken des Login Buttons die Action Login losgeschickt werden. Ein Effekt hört dann darauf ob diese Action aufgerufen wurde. Wenn dem der Fall ist kann der Service der Zuständig ist dafür aufgerufen werden und die Daten können aus dem entsprechenden Backend gezogen werden. Dann löst der Effect eine neue Action aus die wiederum den Reducer triggert. Dieser hört auch auf eine bestimmte Action und wenn diese ausgelöst wird der Store verändert. Dann können schlussendlich in allen Komponenten über den Selektor Daten aus dem Store abgerufen werden. Dies hat den Vorteil das alle Komponenten jederzeit auf demselben Stand sind und es zu keiner Zeit zu doppelten Daten oder Unstimmigkeiten der Daten kommt. Zudem werden die Abfragen minimiert da nicht bei jeder Seite neu das Backend abgefragt werden muss ob die Login Daten überhaupt korrekt sind.

Für einfache Übersicht und Nutzung des Service werden bei unterschiedlichen Aufrufen unterschiedliche Routen gewählt. So kann man anhand der Route immer sehen wo man sich soeben befindet.

Das Frontend läuft mit einer kompilierten Version von Angular in den Docker containern um schneller zu funktionieren und näher am Browser operiert.

## 3.2 Backend

Das Backend unseres Web-Services mit Wettfunktion wurde mithilfe von NodeJS und Express realisiert. Über NodeJS kann man sehr einfach eine Datenbankverbindung zu MySQL-Datenbanken herstellen. Express erlaubt es, relativ einfach eigene Endpunkte für eine REST-API in Javascript zu erstellen. Alle 18 Endpunkte hier vorzustellen wäre wahrscheinlich zu viel des Guten, deswegen belassen wir es bei den 3 Endpunkten von denen in der BPMN Modellierung bei den Punkten 2.1, 2.2 und 2.3 die Rede war.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In der Abbildung lässt sich erkennen, wie sich in NodeJS eine Datenbankverbindung herstellen lässt. Dazu müssen nur Host, User, Passwort sowie Datenbank und ggf. Port angegeben werden und die Datenbankverbindung läuft. Mithilfe dieser ist es möglich, beliebige SQL-Statements in den jeweiligen Endpunkten auszuführen, durch die wir an die Daten aus der Datenbank kommen bzw. auch Daten in die Datenbank schreiben oder updaten können (bspw. beim Cronjob).

### 3.2.1 Endpunkt 2: Ein Match anschauen

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

So sieht ein Endpunkt aus. Aufrufbar wäre er über „localhost:8080/football/match/<id>“.

app.get spezifiziert, dass es sich um einen GET-Request handelt. Alle Spiele in der Datenbank sind über eine MatchID gekennzeichnet. Darüber wollen wir auch Informationen über dieses Spiel erhalten. Der Doppelpunkt vor der id sorgt dafür, dass dieser Bezeichner als Requestparameter behandelt wird – anhand dieser ID können dann im SQL-Statement auch genau die Informationen zu dem Spiel mit der MatchID gefunden werden. Zuvor wird allerdings erst die id geprüft – wenn sie keine Nummer ist, wird ein 400 HTTP Status zurückgegeben. Sollte die Id eine Nummer sein, aber das Ergebnis der Datenbank null sein, so wird ein 204 HTTP Status zurückgegeben. Wenn alles stimmt, wird das Ergebnis (result) in JSON zurückgegeben.

### 3.2.2 Endpunkt 8: Registrieren

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In der Abbildung auf der Seite zuvor erkennt man den Endpunkt fürs Registrieren. Die Requestparameter sind in diesem Fall die E-Mail sowie das Passwort des zu registrierenden Benutzers. Es wird zunächst über eine SELECT Abfrage geprüft, ob die E-Mail bereits in der Datenbank vorhanden ist. Wenn dem nicht so ist, wird über ein INSERT die Daten in die entsprechende Tabelle eingetragen und der Account registriert. Falls eine gleiche E-Mail bereits vorhanden sein sollte, dann wird ein 204 HTTP Status zurückgegeben.

### 3.2.3 Endpunkt 14: Wetten eintragen

Ein Bild, das Text enthält.

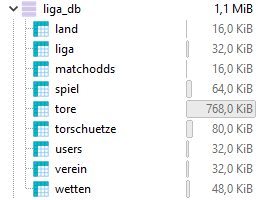
Automatisch generierte Beschreibung

In diesem Endpunkt gibt es relativ viele Requestparameter. Zunächst wird geprüft, ob ein User bereits eine Wette für das Spiel hinterlegt hat. Wenn das nicht der Fall ist, wird eine Wette mit den gesetzten Coins und Toren angelegt. Falls bereits eine Wette existiert, dann wird die bestehende Wette in der Datenbank mit den anderen Werten mit einem UPDATE Statement aktualisiert.

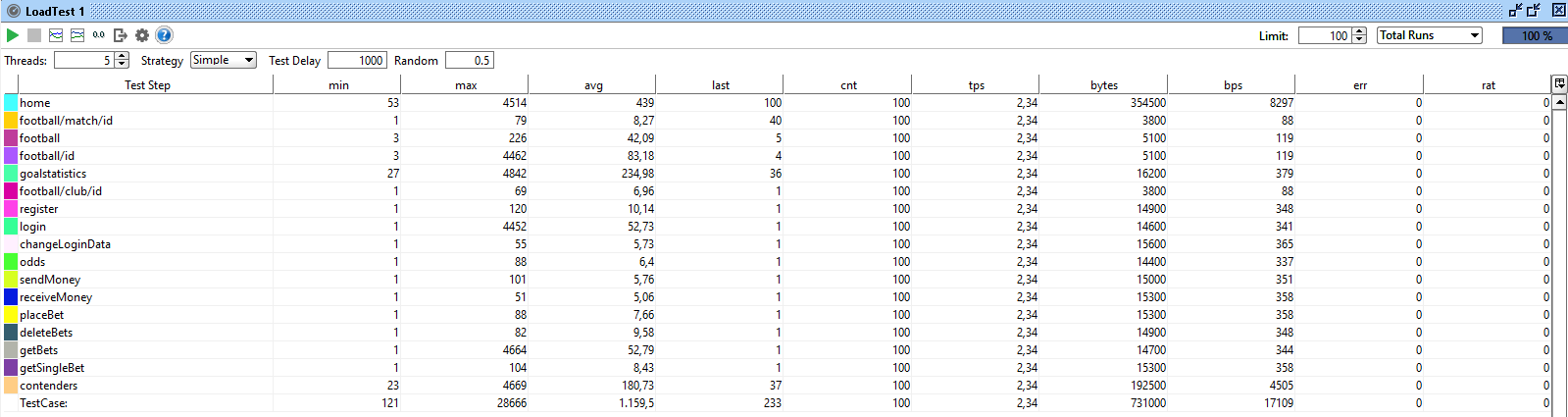
Eine Übersicht über alle Endpunkte finden Sie in der swagger.yaml. Dort sind alle Endpunkte spezifiziert und dokumentiert.

## 3.3 Datenbank

Als Datenbank haben wir eine MySQL-Datenbank verwendet, in welcher wir die Tabellen aus der Abbildung unten benutzt hatten. Diese war notwendig, weil wir auch historische Fußballbundesliga-Daten, sprich von 2015 bis heute, dabei haben wollten. Mit einem einmaligem Seeden, also dem Import der Daten von der OpenligaDB-API war es aber nicht getan, weil sich Ansetzungen, Tore, Torschützen, Wettquoten etc. natürlich ständig ändern. Aus dem Grund haben wir im Backend über node-cron eine Möglichkeit gefunden, einen Cronjob täglich auszuführen, der eben diese Daten in der Datenbank updated, damit diese auf dem neuesten Stand bleibt.



# 4 Performancetest in SoapUI



1. https://www.openligadb.de/api/getmatchdata/39738 [↑](#footnote-ref-1)
2. https://api.the-odds-api.com/v4/sports/soccer\_germany\_bundesliga/odds/?regions=eu&markets=h2h&apiKey=aab6fa5774ec2af0b08b95eef17e9b58%27 [↑](#footnote-ref-2)