**Arbeitstitel:** Kickercheck

**Problemstellung:**

Der kleine Tobi geht jeden morgen voller Elan zum Kickertisch am Campus.Auch in seinen Pausen rennt Tobi fast täglich in den Keller zum Kicker ,nur um festzustellen ,dass dieser belegt ist. Hat der Zug nach Gummersbach einmal Verspätung, so steht Tobi morgens allein am Kicker und hat niemanden zum spielen ☹. Auf diese Weise gehen ihm täglich wertvolle Minuten verloren und seine Stimmung leidet. Beim wöchentlichen Kickerturnier werden Punktetabellen auf Papier geführt, die Anmeldung zum Turnier erfolgt formlos unmittelbar vor dessen Beginn. Um seinen Stand auf der Tabelle abzurufen oder selbst Turniere zu planen muss Tobi sich an den Organisator wenden.

**Projektidee:**

Wäre es nicht praktisch könnte Tobi schon im Zug oder beim Mittagessen checken ob der Kickertisch gerade belegt ist ? Wenn er eine zentrale Anlaufstelle für die Organisation von Matches oder ganzen Turnieren hätte und jederzeit seinen Stand in der Tabelle abrufen könnte? Mit Kickercheck könnte Tobi von seinem Smartphone aus Partner für Matches zwischen den Vorlesungen suchen ohne den Tisch aufzusuchen, Turniere könnten dezentral organisiert und Punktestände mobil eingesehen werden. Über Spielerrankings könnten Matches auf die Fähigkeiten der Spieler abgestimmt werden.

**Dienstgeber**

* Bereitstellen der Belegungssituation des Kickertischs
* Verwaltung von Spielen
* Erstellen von Spielstatistiken
* Verwaltung von Turnieren
* Turnierplan erstellen
* Turnierplan anpassen
* Erstellen von Turnierstatistiken
* Statisitiken bereitstellen
* Erstellen von Spielerstatisiken
* Verwaltung von Herausforderungen
* Spielerranking erstellen
* Saisoninformation bereitstellen

**Dienstnutzer**

* Check der Belegungssituation des Kickertischs
* Belegungssituation ändern (Spielerzahl ändert sich, Jemand fordert den Tisch)
* Ein Spiel anlegen
* Einem Spiel beitreten
* Ein Spiel absagen
* Abruf von Statistiken eines Spiels
* Ein Turnier anlegen
* einem Turnier beitreten
* Turnierteilnahme widerrufen
* Turnierergebnisse abrufen
* Spielerprofil anlegen
* Spieler herausfordern
* Spielerranking abrufen
* Saisoninformation abrufen
* Spielerprofil abrufen

**Abwägungen :**

**Zielsetzung & Guidelines**Der Kickercheck Webservice soll so web-friendly wie möglich gestaltet werden. Bezogen auf seine Architektur bedeutet dies, dass es das Ziel sein muss, Level 3 in Richardsons Maturity Modell zu erreichen und möglichst viele Einschränkungen des REST-Architekturstils zu berücksichtigen. Zu diesem Zweck wurden einige Grundregeln für Entwurf und Implementierung aufgestellt:

* URIs bezeichnen Dinge der Realwelt beziehungsweise Geschäftsobjekte, niemals Operationen
* Ressourcennamen sowie Schnittstellen zum System sollen keinerlei Implementierungsdetails enthalten
* Die Abbildung einer Ressource als Datenobjekt soll möglichst redundanzfrei sein
* Die HTTP Verben sollen gemäß der im Web verbreiteten Semantik implementiert werden
  + GET – Liefert eine Repräsentation einer Ressource, ist safe und idempotent
  + PUT – Aendert den Zustand einer Ressource, deren URI vom Server bereits bereitgestellt wurde. PUT ist idempotent und unterstützt nur absolute , keine relativen Änderungen
  + POST – Erzeugt eine neue Ressource auf dem Server , dieser stellt die URI der neuen Ressource in der Antwort zur Verfügung.
  + DELETE – Ebenfalls idempotent , entfernt eine Ressource auf dem Server. Nach erfolgreichem delete ist die Ressource nicht mehr von außen abrufbar
* Die Kombination einer URI mit einem HTTP Verb muss eine eindeutige Semantik haben
* Der Webservice soll einen einzigen Einstiegspunkt haben, von dort aus kann mit Hilfe von Hypermedia durch die verschiedenen Zustände navigiert werden
* Schließe in die Repräsentation einer Ressource Hypermedia links ein um den Client auf mögliche nächste Zustandsänderungen hinzuweisen
* Für die Implementierung von Hypermedia wird eine Datenkodierung benötigt, die Hypermedia unterstützt ,
* Nutze HTTP Statuscodes als Mittel zur Statuskommunikation mit dem Client ,weise bei Fehlerzuständen auf mögliche nächste Schritte zur Behebung des Fehlerzustands hin
* Verwende so oft wie möglich ETags , so können race conditions vermieden und caching unterstützt werden

**Disclaimer**

Es wird die Annahme getroffen dass alle Nutzer des Kickers die App benutzen und das System nicht missbrauchen. Es wird grundsätzlich von seinem fairen Sportsgeist bei der Austragung der Matches ausgegangen. Maßnahmen zum Schutz gegen Missbrauch oder Angriffen übersteigen den Rahmen dieser Veranstaltung.   
Auch die Authentifizierung eines Nutzers, beispielsweise durch Session Cookies wird zunächst als gegeben betrachtet. Die Entscheidung das System als Webservice im REST-Stil und unter Verwendung des HTTP und seiner Verben zu implementieren ist Teil der Aufgabenstellung.

**Statuscodes**

**Rechte**

Wie bereits erwähnt wird bei der Implementierung ein System zur Authentifizierung vorausgesetzt. Die Authentifizierung wird bei allen PUT, POST und DELETE Operationen vorausgesetzt, lediglich ein GET auf alle Ressourcen ist im unauthentifizierten Zustand möglich.

**Datenhaltung:**Bevor eine geeignete Form der systeminternen Datenhaltung gewählt wurde, mussten einige non-funktionale Anforderungen beleuchtet werden.   
Als solche wurden betrachtet:

* Overhead – Wie groß ist der Unterschied des zu übertragenen Overheads bei Client-Server Interaktionen?
* Flexibilität – In wieweit lässt sich die gewählte Form auf unterschiedliche Anwendungsbereiche ausweiten?
* Skalierbarkeit – Geht man von einigen tausend oder mehr Nutzern aus, die das System gleichzeitig nutzen wollen, welche Alternative bietet bessere Eigenschaften bezüglich Prüfbarkeit , Sicherheit , Performance ? Welche Alternative zeigt die besseren Eigenschaften bezüglich Kompabilität?
* Wartbarkeit – Welche Form der Datenhaltung ist auf Entwickler und Administratorseite besser erweiterbar?
* Hypermedia support – Es muss eine Form der Datenkodierung gewählt werden, die die Angabe von Hyperlinks mit semantischer Zusatzinformation unterstützt.

**Ressource Benutzer:**   
Die Ressource Bentuzer beinhaltet seinen Name, Alter, bevorzugte Position , ein Profilbild und seine Spielstatistik. Diese Entitäten werden nicht als Subressourcen modelliert, weil dies als zu feingranular empfunden wurde. Daraus folgt zwangläufig, dass bei einer Änderung einer dieser Information mittels PUT stets auch die nichtgeänderten Positionen übermittelt werden müssen.

GET: Es muss kein Content Type im request angegeben werden, der Server antwortet mit einer maschinenlesbaren Repräsentation (application/xml) , falls dies ein vom Client gemäß dem gesendeten Accept-header akzeptierter content type ist.

POST: Eine neue Instanz der Benutzerressource wird im System erzeugt. Bei Erfolg wird eine XML-Repräsentation der neu erzeugten Ressource sowie seine serverseitig generierte URI im Location-Feld des headers zurückgesendet.

PUT: Der Client schickt eine vollständige Repräsentation der Ressource mit Content-Type application/xml und Server schickt Content Type 200 wenn Daten erfolgreich geändert wurden (text/html).

DELETE: Kein Content-Type vom Client gesendet. Server antwortet mit Statuscode 202 um zu zeigen dass die Anfrage verstanden wurde. Der Benutzer mit der über das Template spezifizierten ID wird aus dem System entfernt.

**Ressource Match:**

Jedes Match kennt seine Spieler, seinen Austragungsort, Datum und Uhrzeit der Austragung und seinen aktuellen Spielstand. Die Spieler eines Matches werden über Hypermedia Links auf ihre Benutzerressourcen identifiziert , der Standort ebenfalls als Link auf eine Standort-Ressource. Der Spielstand eines Matches hat zwar wenig eigene Attribute , soll aber bei jedem Tor aktualisiert werden , daher wird die Spielstand Information als Subressource des Matches (mit eigenem URI Template) modelliert, sodass Änderungen mittels PUT ohne Übertragung der gesamten Matchressource stattfinden können.

Die Ressource Match soll primär eine Verabredung zum spielen darstellen.   
Es gibt unterschiedliche Ausprägungen einer solchen Verabredung, es muss immer mindestens ein Spieler, ein Austragungsort sowie eine Uhrzeit und ein Datum angegeben werden.   
Datum und Uhrzeit eines Matches werden nicht als Subressourcen modelliert, denn die Änderung dieser Information ändert den Kern der Ressource, daher wird bei einer Änderungsoperation die gesamte XML-Datenstruktur der Matchrepräsentation übertragen. Es wird nicht davon ausgegangen dass diese Information frequent geändert wird.

Es wird sichergestellt, dass ein Match entweder aus 2 oder 4 Mitspielern besteht, damit sein Ergebnis in die Spielerstatistiken eingehen kann.

Wird ein Match in einem Zeitintervall von einer Stunde nicht vollendet (ein Spieler erreicht 6 Punkte) wird es automatisch aus dem System entfernt und geht nicht in die Spielerstatistiken ein.

Sonderfall :   
Ein Match muss vorzeitig abgebrochen werden : Beide Parteien bekommen die Möglichkeit zu bestätigen ,dass das Match nicht gewertet werden soll . Wenn nur eine Partei abbricht wird das Spiel als Niederlage für diese Partei gewertet.

GET: Es muss kein Content Type im request angegeben werden, der Server antwortet mit einer maschinenlesbaren Repräsentation (application/xml) , falls dies ein vom Client gemäß dem gesendeten Accept-header akzeptierter content type ist. Die Repräsentation eines Matches enthält (sofern bereits bekannt) Links auf seine Teilnehmer (Benutzerressourcen) , seinen Austragungsort (Standortressource) seinen aktuellen Spielstand.

PUT: Die Daten eines Matches können bis zu seinem angegebenen Beginnzeitpunkt geändert werden. Teilnehmer ,Standort , Datum und Uhrzeit können so beliebig verändert werden.

POST: Das Anlegen eines Matches erfordert die Angabe seines geplanten Anstoßzeitpunkts (Datum + Uhrzeit) sowie seines Austragungsorts. Es wird eine Instanz der Matchressource ohne Teilnehmer und mit einem Placeholder-Spielstand von X – X erzeugt.

DELETE: Ein Match kann bis zu seinem Beginn abgesagt werden. Diese Absage ist vom Matchersteller auszuführen.

Liegt der Zeitpunkt des Löschens in der Vergangenheit liefert der Server einen 405 – Operation not allowed und liefert im Headerfeld „Allow“ die erlaubten Verben.

**Subressource Spielstand:**

Der aktuelle Spielstand eines Matches wird als dessen Subressource modelliert, da dieser bei jedem Tor aktualisiert werden soll. Damit wird vermieden, dass jedes mal eine vollständige Repräsentation eines Matches an den Server übertragen werden muss, wenn diese Information geändert werden soll. Da ein Spielstand ohne sein zugehöriges Match keine Semantik besäße, ist ein Spielstand nicht als eigenständige Ressource modelliert. Alternativ wäre für diesen Anwendungsfall die Verwendung des Verbes PATCH sinnvoll gewesen, das nur den veränderten Teil eines Datenobjekts übermittelt.

GET- Liefert die Repräsentation eines Spielstands , diese ist der gängigen Punktedarstellung im Fußball nachempfunden Beispiel:Heißt das Match Team1 vs Team2 bedeutet 4:2 , dass Team1 mit 2 Toren in Führung liegt.

PUT – Ändert den Zustand eines Spielstandes , und liefert eine Repräsentation des Zugehörigen Matches mit dem neuen Spielstand zurück

**Ressource Turnier :**

Ein Turnier ist eine geregelte Abfolge von Matches.

Die Attribute eines Turniers sind seine Teilnehmerzahl (immer eine Gerade Anzahl) , sein Typ (Turniere mit Gruppenphase , reine K.O Turniere , evtl. besondere Modi ) ,sein Austragungszeitraum sowie ein Sichtbarkeitsmodifikator um private Turniere zu ermöglichen.

Ein Turnier verweist auf mehrere Match-Ressourcen.

Mittels Querystrings der Form Turnier/{TurnierId}/Match/{MatchId}/Benutzer/{BenutzerId}?select=Name,Profilbild werden Teilinformationen über einen Benutzer in die Repräsentation eines Turniers eingefügt.

Für jedes Turnier werden über die teilnehmenden Benutzer Statistiken wie ein durchschn. Spielerniveau ermittelt. Wenn ein Benutzer ein Turnier anlegen will, spezifiziert er den Turniertyp, die Spielerzahl und den Sichtbarkeitsmodifikator , das Turnier wird dann mit leeren Matches (ohne Teilnehmer) angelegt.

GET: Liefert eine Repräsentation eines Turniers, beinhaltet Links zu allen hinzugefügten Matches.

PUT: Ändern eines Turniers, Teilnehmerzahl , Typ , Sichtbarkeitsmodifikator können verändert werden , solang das Turnier noch nicht begonnen hat

POST: Erzeugt eine Turnierressource im System und liefert gemäß dem spezifizierten Turniertyp eine Repräsentation mit N Slots für seine Matches

**Ressource Lokalitaet:**Eine Lokalität ist ein Ort, der mind. Einen Kickertisch im System zur Verfügung stellen will. Sie ist identifiziert durch ihre Adresse, einen Namen und evtl. eine Beschreibung der Lokalität. Jede Lokalität hat mindestens einen Kickertisch als Subressource. Jeder registrierte Nutzer hat die Möglichkeit eine Lokalität im System anzulegen.

Die Ressource Lokalitaet wurde nicht als einfache Adresse eines Kickertischs modelliert, da der Charakter des Tischkickersports auf Geselligkeit unter den Spielern beruht, und daher zusätzliche Information über das Umfeld des Kickertisches, beispielsweise in einer Kneipe, interessant ist.Diese Zusatzinformation wurde semantisch nicht zu der Ressource Kickertisch gezaehlt, was die Einführung einer Zusatzressource notwendig machte.

GET: Es muss kein Content Type im request angegeben werden, der Server antwortet mit einer maschinenlesbaren Repräsentation (application/xml) , falls dies ein vom Client gemäß dem gesendeten Accept-header akzeptierter content type ist. Die Repräsentation einer Lokalitaet enthält Hyperlinks auf alle von ihr gestellten Kickertischressourcen.

PUT: Adresse, Name oder Beschreibung einer Lokalitaet können verändert werden.

POST: Erzeugt eine neue Lokalitaet im System mit den Attributen Adresse,Name und Beschreibung. Die Subressource Kickertisch wird zunächst nicht erzeugt , sondern mit einem nachtreaglichen POST auf der Kickertischressource einem Standort zugeordnet.

Ein erfolgreiches POST liefert eine Repräsentation der angelegten Ressource zurück

DELETE: Der Benutzer, der eine Lokalitaet angelegt hat verfügt über das Recht sie aus dem System zu löschen

**Ressource Tisch:**

Ein Kickertisch ist eine Subressource einer Lokalität, seine Attribute sind sein Hersteller / Typ sowie ein kurzer Kommentar zu dessen Zustand.

PUT : Aendert das Attribut „Zustand“ eines Kickertischs , der Hersteller und Typ des Tisches kann nicht verändert werden

POST: Erzeugt eine Kickertischressource im System und fuegt sie der Lokalitaet mit der angegebenen LokalitaetId hinzu. Liefert eine Repräsentation der Lokalität , der der Tisch hinzugefügt wurde zurück

DELETE: Entfernt einen Kickertisch aus dem System und der Repräsentation der Lokalitaet zu der er gehört.

Template : /Lokalitaet/{LokalitaetId}/Kickertisch

**Ressource Account** (Wird zunächst ausgespart)  
Die Ressource Account realisiert das Rechtesystem und beinhaltet lediglich eine eindeutige Email-Adresse sowie ein dazugehöriges Passwort. Ein Account verweist auf eine Benutzerressource

E-Mail und Passwort sind keine Sub-Ressourcen, die getrennt voneinander identifiziert werden.

GET: Validieren ob gültige E-Mail und Passwort, daher vom Client als XML übertragen (Content-Type). Server schickt Dokument in text/html als Content-Type zurück mit Statuscode 200. Wenn der Client im Accept Header einen nicht unterstützten Content-Type erfodert schickt der Server einen Statuscode 406 (Not Acceptable) zurück.

POST: Server schickt Status Code 201 zurück wenn die Eingaben erfolgreich validiert wurden. Server schickt Status Code 400 wenn die Daten nicht vollständig waren.

PUT: Server schickt Status Code 204 wenn die Accountdaten erfolgreich modifiziert wurden. Server schickt Status Code 403 wenn die Accountdaten nicht modifiziert werden dürfen.

DELETE: Kein Content-Type vom Client gesendet. Server antwortet mit Content Type 202 um zu zeigen dass die Anfrage verstanden wurde. Server antwortet mit Content Type 401 wenn man nicht eingeloggt ist.

**Link Relations**

Benutzer ()   
Match ([0..4] Benutzer/Teilnehmer , Lokalität, Spielstand)

Lokalität ([1..n] Kickertische)

Turnier (Matches)

**Processing Model**

Jeder Tisch kennt seine Belegungssituation (generiert aus der Matchressource die dort stattfindet) kann gefordert werden, und verweist auf alle seine Forderungen.

Eine Forderung ist eine Subressource des Kickertischs und beinhaltet die Information wann die Forderung gestellt wurde sowie Links zu den fordernden Spielern. Es ist möglich als Einzelperson zu fordern , sowie als Team. Einzelpersonen werden dynamisch mit einem Mitspieler gematcht , ohne die Forderungsreihenfolge zu missachten.

Sonderfall 1: Am Tisch findet ein 1gg1 statt, eine Einzelperson fordert , der Verlierer des 1gg1 muss weichen.

Sonderfall 2 : Es findet ein 2gg2 statt , eine Einzelperson fordert , nach der Niederlage weichen 2 Spieler. Bei 3 Spielern am Tisch wird stets auf ein 1 gg 1 zurückgefallen. Die Entscheidung wer aus dem gewinnenden Team weichen muss wird nach dem anmeldenden Benutzer gefällt.

Sonderfall 3: Am Tisch findet ein 2gg2 statt , eine Einzelperson fordert, ein Spieler aus dem Verliererteam muss weichen , das Team wird mit der Einzelperson aufgefüllt.   
  
Sonderfall 4 : Am Tisch findet ein 2gg2 statt, eine Einzelperson fordert , es existieren weitere Forderungen von Zweierteams, die hinter der Einzelperson gefordert haben.

TODO:   
  
- Überall wo req content type = xml , gib auch xml zurück

- Liefere immer eine Repräsentation einer angelegten Ressource   
- Eine Repräsentation enthält keine Bestandteile der repräsentation anderer Ressourcen