**Web-basierte Anwendungen 2:**

**Verteilte Systeme**

Fachhochschule Köln Campus Gummersbach

Fakultät für Informatik und Ingenieurwissenschaften

**Phase 2 - Projekt "Social Ticker"**

**Autoren**

Dario Vizzaccaro 11085033

Benedikt Wurth 11084022

**Inhaltsverzeichnis**

**Einleitung**

Im Rahmen der Veranstalltung Webbasierte Anwendungen 2 sollte ein Projekt über mehrere Monate verwirklicht werden. Dieses Projekt befasst sich mit hauptsächlich mit der Kommunikation zwischen Server und Client und wie Daten in Form von xml-Dateien transportiert werden. Zusätzlich sollte der Transport auf verschiedene Weisen - also synchron bzw. asynchron - implementiert werden. Um das notwendige Wissen zu bekommen, fanden neben dem Projekt Vorlesungen statt, die auf das Projekt vorbereiten sollten und zeitnah die bevorstehenden Hindernisse aufklären sollten.

**Projektidee**

Nach dem ersten Termin entstanden zahlreiche Ideen. Die erste Idee war ein Notizzettel, der für mehrere Personen geteilt werden kann und es somit ermöglicht z.B. seine Einkaufsliste oder gemeinsame anstehende Termin für mehrere Personen zugänglich zu machen, oder eine Applikation zum verwalten von seinem geliehenen Geld. Es kann öfters mal vorkommen dass man Geld an seine Freunde verleiht und um da den überblick zu behalten wäre es interessant eine App zu haben. Da es den ersten Ideen aber am Funktionsumfang und an Innovation mangelte, entschlossen wir uns für die Idee einen Sportticker zu entwickeln, welcher es ermöglicht Nutzern einen eigenen Ticker anzulegen für ein Event seiner Wahl und somit seine Freunde immer auf den aktuellen Stand zu halten. Wenn jemand sich also zum Beispiel ein Handballspiel anguckt und möchte dass auch andere ihm beim zugucken folgen, kann er einfach dieses Ereignis anlegen und selber Kommentare zu dem Spiel veröffentlichen, damit jeder weiß was gerade auf dem Spielfeld passiert.

Als Grundfunktion wird das Erstellen von Events wichtig sein. Ein Nutzer kann für eine beliebige Sportart einen Liveticker ( Event ) erstellen. Er kommentiert dann dieses Event und aktualisiert den Spielstand und die Ereignisse. Nutzer können dem Event dann beitreten, seine Beiträge verfolgen und zu diesem Kommentare verfassen bzw. diskutieren. Der Ticker kann in Echtzeit bewertet werden, damit weitere Nutzer sehen können, ob der Ticker es wert ist, verfolgt zu werden.

Der Reiz an dem Erstellen eines Tickers ist, dass ein Nutzer seine eigenen Fans bekommen kann, die zusammen mit ihm Sportereignisse miterleben und Meinungen austauschen. Der Ersteller eines Tickers ist dabei der "Leiter". Er schreibt die wichtigen Ereignisse und seine Beiträge werden kommentiert bzw. von anderen Nutzern diskutiert.

Die App bietet eine übersichtliche Listung der Ereignisse und unterstützt die Kommunikation zwischen den Usern. Es ist zusätzlich möglich mehrere Ticker gleichzeitig in einem Fenster zu verfolgen. Wenn man z.B. Fan von zwei Teams ist, die an verschiedenen Spielen teilnehmen, will man trotzdem beide gleichzeitig erleben. Auch kann man so mehrere Ticker zum gleichen Spiel folgen und bekommt so mehr Infos und eine vielschichtige Sicht auf Geschehnisse.

**Meilenstein 1:**

**Projektspezifisches XML Schema**

Um die Informationen zu verwalten benötigt man XML-Dateien. Extensible Markup Language kurz XML ist eine Auszeichnungssprache, die es ermöglicht Informationen bzw Text zu strukturieren. Eine XML Datei wird durch passendes XML Schema (XSD) beschrieben. In der XSD werden die Typen des XML bestimmt und durch passende Restriktionen begrenzt. Um also später Dateien zwischen Server und Client zu verschicken, bietet XML die richtige Basis.[[1]](#footnote-1)

Alle Daten solten in drei XML-Dateien ausgelagert. Eine Datei enthält alle Daten der Nutzer. Hier werden die wichtigen Daten wie Username, Vorname, Nachname, Geburtsdatum, Anzahl erstellte Events und Anzahl geposteter Beiträge gespeichert. Die zweite Datei ist für die Events. Hier werden alles Grund-Informationen eines Events gespeichert. Eventname, Eventdatum, Eventadmin, Eventtyp, Eventdauer und Bewertung werden hier für alle Events abgelegt. Auch hier vergibt das System zusätzlich noch eine EventID zur eindeutigen Zuweisung. Die letzte Datei ist für den kompletten Content, also den Beiträgen und Kommentaren zuständig. Man hätte den Content und das Event zwar in ein XML-Dokument zusammenfassen können. Jedoch verändert sich der Content ständig und schnell, wobei die Event-Informationen statisch sind und nach anlegen nicht mehr verändert werden. Es macht somit Sinn den ständig verändernden Teil der Events auf eine extra Datei auszulagern. Dadurch, dass der Content ebenfalls die EventID bekommt, ist es kein Problem dem Event den zugehörigen Content zuzuordnen.

**Userprofiles:**

Ein Nutzer des Dienstes muss sich durch ein Profil eindeutig identifizieren, falls er Events oder Kommentare veröffentlichen will. Der Nutzer identifiziert sich durch seinen Usernamen. Zusätzlich legt das System automatisch eine UserID für den einfacheren Umgang mit den Usern an. Die UserID wird später benötigt, um die Events und Beiträge eindeutig den Nutzern zuzuordnen.

Der User kann dann noch weitere Angaben machen, um von seinen Freunden erkannt zu werden. Vor-, Nachnamen und Geburtstag sind optionale Angaben, falls der User diese nicht veröffentlichen will. Ein User kann Freunde hinzufügen, um auf dem aktuellsten Stand zu bleiben, was seine Freunde so machen.

Die Anzahl erstellter Events und geposteter Beiträge werden automatisch im Profil hinterlegt. Hier kann jeder User sehen, wie aktiv man selbst oder andere im Social Ticker sind.

Um alle Nutzer in einer Datei abzuspeichern werden alle User in einer „Userlist“ abgespeichert. Die Userlist besteht dann aus allen Usern, die am Social Ticker teilnehmen. Wichtig ist die Angabe, dass in dem Element mehrere Elemente auftauchen können hierfür ist die minOccurs und maxOccurs Angabe. Diese gibt an, wie oft ein Element mindestens auftauchen muss oder maximal auftauchen kann.[[2]](#footnote-2)

*<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">*

*<xs:element name="userlist">*

*<xs:complexType>*

*<xs:sequence>*

*<xs:element ref="user" maxOccurs="unbounded" minOccurs="0" />*

*</xs:sequence>*

*</xs:complexType>*

*</xs:element>*

**simple-types:**

Simple-Types sind dafür da die einzelnen Elemente zu typisieren und zu benennen. Wichtig ist es Angaben zu machen von welchem Typ das Element sein soll. Typische Angaben sind „xs:string“, „xs:integer“, „xs:date“ oder „xs:time“. Aus den oben genannten Angaben entstehen folgende simple-types.

* Username (String)
* Vorname (String)
* Nachname (String)
* Geburtsdatum (Date)
* erstellte Events (Decimal)
* gepostete Beträge (Decimal)

Beispiel:

*<!-- Username -->*

*<xs:element name="username“ type="xs:string"/>*

**complexe-types:**

Complexe-Types sind dafür da mehrere Simple-Types in einem Element zusammenzufassen. Da User aus mehreren Informationen bestehen und man nicht alle Informationen als einen String abspeichert, werden diese auf mehrere Elemente aufgeteilt. Nach den oben genannten Vorgaben enstehen also folgende complexe-types.

* User (Username, Vorname, Nachname, Geburtsdatum, Geschlecht, erstellte Events, gepostete Beträge, Folger, Favoriten)\*
* Folger (Username)\*
* Favoriten (Username)\*

Beispiel:

*<!--Vollstaendige Userdaten -->*

*<xs:element name="user">*

*<xs:complexType>*

*<xs:sequence>*

*<xs:element ref="username"/>*

*<xs:element ref="vorname"/>*

*<xs:element ref="name"/>*

*<xs:element ref="gender"/>*

*<xs:element ref="geburtsdatum"/>*

*<xs:element ref="land"/>*

*<xs:element ref="stadt"/>*

*<xs:element ref="anzEvents"/>*

*<xs:element ref="beitraege"/>*

*<xs:element ref="favoriten"/>*

*<xs:element ref="folger"/>*

*</xs:sequence>*

*<!-- Bei Registrierung automatisch generierte einzigartige ID -->*

*<xs:attribute type="xs:positiveInteger" name="userID"/>*

*</xs:complexType>*

*</xs:element>*

Wie man an dem Beispiel sieht, ist die UserID als „attribute“ initialisiert, dies macht Sinn, da die ID generisch ist und sie nur zu Verwaltungszwecken genutzt wird.

**Restriktionen:**

Restriktionen werden benötigt, um die Elemente in einem passenden Rahmen angeben zu können. Es macht z.B. keinen Sinn einen Usernamen anzulegen, der 100 Buchstaben lang ist, deswegen werden die Elemente in ihrer Größe begrenzt. Restriktionen können auch dazu dienen, ein Element in der Auswahlmöglichkeit zu begrenzen. Das Geschlecht kann nämlich nur „männlich“ oder „weiblich“ sein als macht es hier Sinn, dies auch als Restriktion anzugeben

* Username min.3, max.15 Zeichen
* Vorname min.2, max. 20 Zeichen
* Nachname min.2, max. 20 Zeichen
* Geschlecht ist „Male“ oder „Female“
* Geburtsdatum zwischen zwischen 1900 und 2010
* Land ist unter „Germany“,“Italy“, „France“ und weiteren ausgewählten Ländern

Beispiel:

*<!-- Username -->*

*<xs:element name="username">*

*<xs:simpleType>*

*<xs:restriction base="xs:string">*

*<xs:minLength value="3"/>*

*<xs:maxLength value="15"/>*

*</xs:restriction>*

*</xs:simpleType>*

*</xs:element>*

<!-- Geschlecht -->

<xs:element name=*"gender"*>

<xs:simpleType>

<xs:restriction base=*"xs:string"*>

<xs:enumeration value=*"Male"*/>

<xs:enumeration value=*"Female"*/>

</xs:restriction>

</xs:simpleType>

</xs:element>

**Events**

Die Events sind das Herzstück unseres Dienstes. Es können hier Events (wie zB ein Fußball- oder Handballspiel) erstellt und verwaltet werden. Ein Event benötigt natürlich eine eindeutige Kennung. Der Name wäre hierbei nicht in jedem Fall eindeutig. Deswegen wird eine ID zur einfacheren Verwaltung erstellt. Eventname gibt an um welches Thema der Ticker überhaupt geht. Der Eventadmin ist sofort der User, welcher das Event erstellt hat. Auch interessant für den User ist es, um welchen Typ es von Event handelt. Also ob es ein Football-, Faustball- oder doch Golfspiel ist. Ein Event kann von anderen Usern bewertet werden. Die Wertung wird abgespeichert und automatisch verrechnet.

**simple-types:**

* Eventname (String)
* Eventdatum (Date)
* Eventadmin (Username)
* Eventtyp (String)
* Eventstart (time)
* Eventende (time)
* Bewertung (positiveInteger)

**complexe-types:**

* Eventlist(Event\*)
* Event (EventID, Eventname, UserID, Eventtyp, Eventstart, Eventende, Eventdauer, Bewertung)

**Restriktionen:**

* Eventname min. 5, max. 100 Zeichen
* Eventbeschreibung optional min. 3, max. 250 Zeichen
* Eventtyp min. 3, max. 100 Zeichen
* Eventbewertung optional <= 10

**Eventcontent**

Um alle Beiträge(vom Admin) und Kommentare(von Usern) abzuspeichern, wird der Eventcontent benötigt .Zentral bei den Beiträgen des Events ist das "Soziale". Hierbei sind die Beiträge des Admins der Dreh- und Angelpunkt des Events. Andere User können dann zusätzlich noch unter den Beiträgen des Admins Kommentare setzen und diskutieren, ob sie die Geschehnisse genau so interpretieren oder einen anderen Standpunkt besitzen. Wichtig dabei ist auch, dass dem Admin eine Möglichkeit geboten wird, den aktuellen Spielstand zu veröffentlichen.

**simple-types:**

* TickerBeitrag (String)
* KommentarText (String)
* Heim (String)
* Gast (String)

**complexe-types:**

* EventContentList(Eventcontent\*)
* EventContent(aktuellerStand, eventID ,TickerBeitrag\*)
* TickerBeitrag (Text,Kommentar\*)
* Kommentar(Username,KommentarText)
* aktuellerStand(Heim,Gast)

**Restriktionen:**

* TickerBeitrag min. 3 Zeichen
* KommentarText min. 3 Zeichen

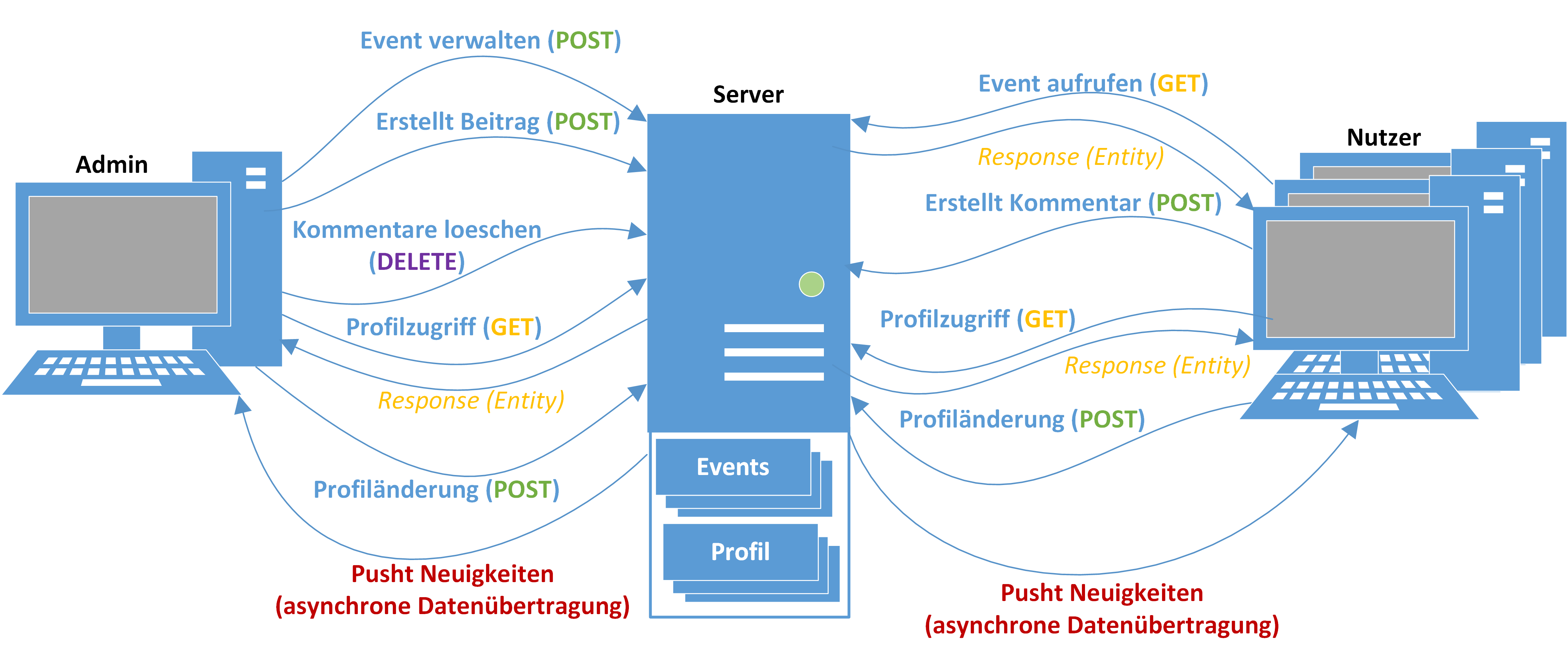
**Kommunikationsabläufe und Interaktion**

Im Grunde genommen ist erst einmal jeder Nutzer gleich und ihm stehen die gleichen Funktionen zur Verfügung. Jeder Nutzer kann auf sein eigenes Profil zugreifen und dieses auch verändern. Zusätzlich kann er auch andere Profile aufrufen und sich Freunde anschauen. Die Interaktion mit dem Server unterscheidet sich erst wenn man ein Event erstellt bzw. ein Event aufruft. Als Administrator (Ersteller eines Events) hat man einen erweiterten Funktionsumfang. Als allererstes sollte man für sein Event Informationen anlegen ("Event verwalten"). Wenn man alles soweit angelegt hat, kann man Beiträge verfassen und diese dann veröffentlichen. Sie werden chronologisch aufgelistet. Andere Nutzer, die das Event aufrufen können dann die erstellten Beiträge kommentieren. Als Admin ist es möglich Kommentare anderer Nutzer zu löschen. Als Nutzer (Aufrufer eines Events) kann man sich eine Liste aller Events anzeigen lassen. Dort wählt man dann ein Event auf, welches einen interessiert. In einem Event angekommen, sieht man alle aktuellen Beiträge und Kommentare. Man kann sich nun entscheiden, ob man selbst ein Kommentar verfassen will oder ob man erst einmal den Ticker verfolgt, bis man seine eigene Meinung veröffentlicht.

**Synchrone Datenübertragung**

­

**Asynchrone Datenübertragung**



**Meilenstein 2:**

**Ressourcen ,die Semantik der HTTP-Operationen**

Die Daten sollten auf drei Typen ausgelagert werden. Jeder Ticker selbst wird in einem Event gespeichert. Der zugehörige "Content" also die Beträge des Admins und die Kommentare anderer User werden dabei in eine zweite Datei gespeichert. Die Aufteilung von Content und Event ist sinnvoll, da der Content dynamische Daten enthält. Diese werden oft verändert, wobei das Event selbst mit den zugehörigen Informationen meist einmal angelegt und nicht mehr verändert wird.

Die Profile unserer Nutzer werden in einer dritten Datei gespeichert. Bei uns entstehen somit drei Ressourcen - Event, Userprofiles, Eventcontent. Aus diesen Ressourcen ergeben sich folgende Operationen:

**Admin (Ersteller eines Events):**

Ein Admin hat andere Rechte bzw. andere Funktionen als ein normaler Nutzer, der auf einen Ticker zugreift. Er hat die volle Kontrolle, kann Spielstände aktualisieren, Beiträge löschen und überhaupt den Ticker erst mit wichtigen Informationen - den Beiträgen - füllen. Dem Admin stehen zusätzlich die Funktionen der User zur Verfügung.

| Operation | Beschreibung | | ----------------- | ------------ | | POST /events | Admin erstellt ein Event. | | PUT /events/$eID | Ändern bzw aktualisieren eines Events. | | POST /events/$eID/EventContent | Erstellt einen Beitrag im Ticker. | | DELETE /events/$eID | Löscht ein als Admin erstelltes Event. | | DELETE /events/$eID/EventContent/beitrag/$tBID | Löscht Beitrag in seinem Event. |

**User:**

Als User hat man die Standard Funktionen. Die URIs wurden wie folgt gewählt | Operation | Beschreibung | | ----------------- | ------------ | | GET /events | Bekommt Events zurück. | | GET /events/$eID | Bekommt alle Informationen eines Events. | | GET /events/$eID/eventcontent | Bekommt gesammten Content eines Events. |

| POST /events/$eID/eventcontent/beitrag/$tBID | Erstellt einen Kommentar zu einem Beitrag im Ticker. | | GET /users/$uID | Kann sich eigenes oder fremde Userprofiles anzeigen lassen. |



| GET /users | Gibt alle User wieder. | | PUT /users/$uID | Kann sein eigenes Userprofiles bearbeiten. |

**RESTful Webservice Implementierung der Operationen**

Um auf die xml-Daten zuzugreifen wird Marshalling und Unmarshalling verwendet. Marshalling dient dazu aus einer angelegten Datei, xml-Daten auszulesen. Danach können Daten angehängt, verändert oder gelöscht werden und anschließend durchs Unmarshalling wieder abgespeichert werden.

**PathParams und QueryParams**

Mit PathParams kann man auf einen bestimmten Teil der Daten zugreifen. So ist es möglich ein bestimmtes Event bzw einen bestimmten User aufzurufen. Mit QueryParams lassen sich Ergebnisse filtern. Sinnvolle Filterung wäre z.B. User nach Namen oder Land zu filtern. Genau so macht es Sinn die Events nach Namen des Events zu filtern. Als Beispiel wäre ein Nutzer der alle Spiele von Bayern haben möchte und als Filter den Namen Bayern eingibt.

**Konzeption + XMPP Server einrichten Leafs (Topics)**

Jedes Event ist Abonnierbar. Sobald man das Event joint bekommt man als erstes alle Nachrichten als GET. Wenn es dann aufgerufen ist, werden neue Benachrichtigungen per asynchrone Benachrichtungen verschickt. Ein User kann selbst neue Kommentare schreiben oder von anderen welche zugestellt bekommen.

**Publisher**

Im Grunde benommen ist jeder Publisher, der sich in einem Event befindet. Als User kann man Kommentare erstellen und diese werden asynchron an andere gepusht. Der Admnis selbst ist natürlich auch Publisher. Er erstellt Beiträge zu seinem Event welche alle zugestellt bekommen.

**Subscriber**

Genauso wie jeder Publisher sein kann, ist auch jeder Subscriber der sich in einem Event befindet. Sobald man einem Event beitritt kriegt man alle neuen Beiträge und Kommentare asynchron.

**Zu übertragene Daten**

Die übertragbaren Daten sind die Beiträge eines Admins und die Kommentare der User. Im Grunde genommen ist der Ticker nicht viel anders als ein Chat mit besonderer Formatierung und Featuers wie Aktueller Stand und Bewertungen.

**XMPP - Client Client - Entwicklung**



1. Buch: REST und HTTP, Stefan Tilkov, 2011 s.84ff. [↑](#footnote-ref-1)
2. http://www.w3schools.com/schema/schema\_complex\_indicators.asp [↑](#footnote-ref-2)