Dokumentation Phase 2
WBA 2

Oliver Thesenvitz, Dennis Jaeger

23. Juni 2013

Inhaltsverzeichnis

1 Projektvorfeld

Im Workshop zum Modul "Webbsasierte Anwendungen" sollen die in der Vorlesung vermittelten Kenntnisse in der Praxis angewendet und vertieft werden. Dazu wird in mehreren Schritten eine verteilte Anwendung auf Basis eines eigens definierten Konzepts entworfen. Die erforderlichen Daten werden in XML-Dateien abgelegt und mit einer auf dem RESTful-Webservice-Stil behandelt. Um die Kommunikation und eine asynchrone Datenübermittlung zu ermöglichen wird im Anschluss ein XMPP-Server installiert und die erforderlichen Funktionen programmiert. Abschließend wird eine Benutzeroberfläche erstellt, um die zuvor entwickelten Funktionen gebrauchstauglich nutzen zu können.

2 Projektidee

Unsere Idee war es eine Fussball-Applikation zu entwickeln. Jeder Fussball begeisterte Mensch soll die Möglichkeit bekommen alles über seinen Lieblingsverein zu erfahren, während eines Livetickers mitzufiebern oder einfach über Pushmitteilungen zu erfahren, falls ein Tor geschossen wurde.

3 Konzeptioneller Meilenstein - Kommunikationsabläufe und Interaktionen

3.1 Ressourcen

Im Folgenden werden die in unserem Projekt identifizierten Ressourcen erläutert.

Tabelle

In „Tabelle” wird die aktuelle Rangliste aller Mannschaften abgelegt. Diese wird nach jedem Spieltag aktualisiert. Die Mannschaften werden absteigend anhand der erspielten Punkte geordnet. Neben der Reihenfolge, den erspielten Punkten sowie Namen der Mannschaften werden außerdem die geschossenen Tore sowie Gegentore und die daraus resultierende Tordifferenz angezeigt.

Spiel

Die aktuelle Begegnung einer Heim- gegen eine Gastmannschaft. Im Verlauf des Spiels werden optional die erzielten Tore bzw. das Endergebnis hinzugefügt.

Tor

Angabe eines erzielten Treffers in einem Fussball-Spiel. Zugehörig werden der Zeitpunkt des Tores (Spielminute) sowie der Torschütze gespeichert.

Spieltag

Es werden die zu diesem Spieltag zugehörigen Spielbegegungen (Spiel), bestehend aus Heimmannschaft, Gastmannschaft und Ergebnis, abgelegt.

Liveticker

Diese Ressource besteht aus Echtzeit-Kommentaren zu einem gegenwärtig laufenden Spiel. Diese werden als String (Text) von einem Kommentator eingegeben. Ein Kommentar hat folgende Bestandteile: Name der betroffenen Mannschaft sowie Spielereignis (ein Tor oder das Ende einer Begegnung).

Mannschaft

In der Ressource Mannschaft werden die verschiedenen Fussball-Spieler zu einem Fussball-Verein zusammengefasst. Alternativ dazu ist es möglich eine Ressource für jeden einzelnen Spieler zu entwerfen. Dadurch wären konkretere Angaben zu den einzelnen Spielern, beispielsweise die Spielerposition oder die Trikotnummer, realisierbar gewesen. Dieser hohe Detailierungsgrad ist aber in unserem Kontext nicht notwendig und würde nur die Komplexität der Anwendung massiv erhöhen. Aus diesem Grund entscheiden wir uns für die Abbildung der Spieler in der Ressource Mannschaft.

Tippspiel

Die Benutzer der Anwendung können bis zum Anpfiff einer Spielbegegnung einen Tipp abgeben, wie das Spiel ihrer Meinung nach enden könnte (Spielergebnis). Nach Spielende wird der zuvor abgegebene Tipp mit dem realen Endergebnis verglichen. Eine Liste aller Benutzer die den richtigen Tipp abgegeben haben kann nach Beendigung des Spiels abgefragt werden.

3.2 Synchrone Interaktionen

- Aktuelle Tabelle abrufen
- Beliebig gewähltes Spiel abrufen
- Spieltag (Übersicht aller Spiele) abrufen
- Liveticker eines Spiels starten
- Der zu folgenden Mannschaft auswählen
- Tipp für ein Tippspiel festlegen
- Tipp für ein Tippspiel ändern
- Tipp für ein Tippspiel löschen
- Eigenen abgegebenen Tipp mit dem realen Endergebnis eines Spiels vergleichen

3.3 Asynchrone Interaktionen

- Kommentare vom Liveticker abrufen
- Toralarm bei einem geschossenem Tor erhalten

3.4 Übermittelte Daten

Tabelle

- Mannschaftsnamen
- Aktuelle Punktzahlen aller Mannschaften
- Platzierung der Mannschaften
- Erzielte Tore
- Gegentore
- Tordifferenz (Differenz aus erzielten Toren und Gegentoren)

Spiel

- Ergebnis
- Tore
- Torschützen
- Heimmannschaft
- Auswärtsmannschaft

Tor

- Torschütze
- Mannschaft des Schützen
- Minute

Spieltag

- Spieltagsnummer
- Heimmannschaften
- Auswärtsmannschaften
- Ergebnisse

Liveticker

- Heimmannschaft
- Auswärtsmannschaft
- Aktuelles Ergebnis
- Kommentare

Livebenachrichtigung beim Folgen einer Mannschaft

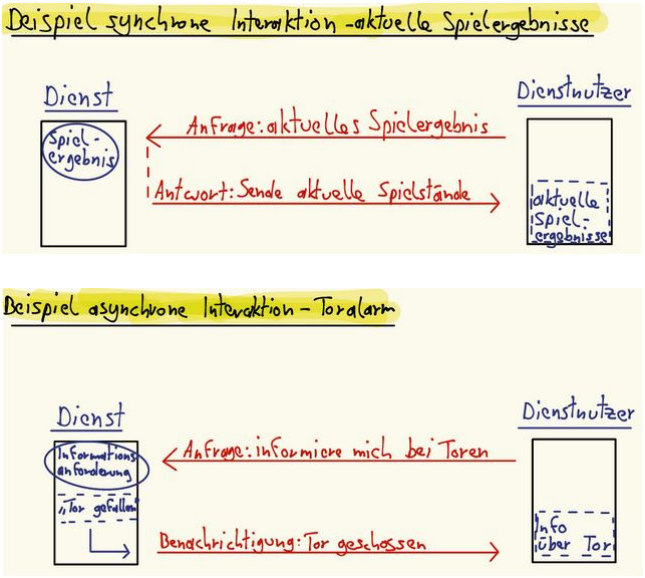
- Spiel
- Tor (Torschütze / Minute)

Tippspiel

- Name des Benutzers
- Eigener Tipp
- Reales Ergebnis
- Spiel

3.5 Skizze - Interaktionen

In der folgenden Skizze wurde anhand der Interaktionen „Spielergebnis abfragen" bzw. „Toralarm" jeweils eine synchrone bzw. asynchrone Interaktion unserer Anwendung exemplarisch dargestellt. Da sich die synchronen bzw. asynchronen Interaktionen sehr ähneln, genügt hier sowohl aus unserer als auch aus Sicht der Betreuer jeweils eine Abbildung jeder Funktionsart um die Interaktionen zu verdeutlichen.



4 XML und XML-Schemata

Für die im Projekt notwendigen XML-Dateien und der zugehörigen Schemata bedarf es einer Festlegung der Sichtweise. In der ersten Vorgehensweise können die XML-Dateien möglichst semantisch und frei von Redundanzen entwickelt werden. Der Vorteil dieser Auslegung ist durch die nicht vorhandenen Redundanzen geringerer Speicherplatzbedarf. Des Weiteren ist durch die atomare Einteilung die jeweiligen Elemente alles etwas kompakter und dadurch einfacher zu

lesen. Alternativ dazu kann die XML-Struktur anhand der späteren Programmlogik entwickelt werden. Dazu werden Überlegungen wie „welche Daten werden bei welchen Funktionen benötigt" in den Vordergrund gestellt. Somit liegen, ähnlich wie bei der XML-Struktur in Phase 1 des Praktikums, alle relevanten Daten möglichst lokal beieinander. Die Vorteile dieser Alternative sind eine einfachere Nachvollziehbarkeit des Gesamtprojektes in der entwickelten Software sowie eine verbesserte Maschinenlesbarkeit. Wir werden im Folgenden mit der erwähnten Alternative (Möglichkeit 2) vorgehen und unsere XML-Struktur ähnlich der ersten Phase des Workshops mit möglichst vollständigen und größeren XML-Daten vorgehen. Die Entscheidung dazu beruht auf der besseren Nachvollziehbarkeit im Projektverlauf, sowie der Übersichtlichkeit in den XML-Dateien selber. In den folgenden Punkten wird die Umsetzung der benötigten Ressourcen in XML-Dateien sowie die zugehörigen Schemata erläutert. Des Weiteren werden die verworfenen Ressourcen erwähnt und die tatsächlich umgesetzten XML-Datein inkl. ihrer Schemata visualisiert.

4.1 Nicht weiter berücksichtigte Ressourcen

Tabelle

Da unsere Anwendung nur mit Beispieldaten gefüllt und nicht auf eine Bundesliga-Datenbank zugreifen wird haben wir uns gegen eine Darstellung der Ligatabelle entschieden. Diese würde keinen erkennbaren Mehrwert für das Projekt bedeuten und allein die Komplexität der Anwendung erhöhen. Eine Befüllung der Tabelle mit Beispieldaten würde den Datenumfang und damit die Komplexität erhöhen.

Spiel und Tor

Die Ressourcen Spiel und Tor werden abstrahiert und in der Ressource Spieltag abgebildet. Dies ist für die weitere Entwicklung der Anwendung aus unserer Sicht sinnvoller. Eine einzelne Implementierung der Ressourcen ergibt in unserem Kontext für uns keinen Sinn oder Mehrwert für das System. Mit der Entwicklung einzelner XML-Schemata für die Ressourcen Spiel und Tor würde lediglich die Komplexität erhöht werden.

4.2 Geplante Umsetzung von Ressourcen in XML-Dateien

Spieltag

Die Ressource Spieltag wird in XML umgesetzt, um das synchrone Abfragen von aktuellen Spieltag-Ergebnissen zu realisieren. Dabei beziehen wir uns, da wir nur einen Beispiel-Datensatz verwenden, lediglich auf den aktuellen bzw. letzten Spieltag und nicht auf vorhergehende Spieltage. Im betreffenden XML-Schema werden u. A. Heim- und Gastmannschaft abgelegt. Diese Angaben könnten alternativ in einzelne XML-Files abgelegt werden. Eine zentrale Zusammenfassung in einer XML-Datei erzielt hier aber eine bessere Nachvollziehbarkeit. Eine Dezentralisierung würde hingegen eine größere Komplexität bedeuten. Auch werden hier zu Gunsten der Lesbarkeit und Nachvollziehbarkeit eventuelle Redundanzen in Kauf genommen. Da an jedem Spieltag alle Mannschaften antreten müssen finden bedingt durch die Anzahl der Mannschaften genau 9 Spiele statt.

Diese Restriktion ist im XML-Schema abgebildet. Da in jeder Saison genau 34 Spieltage stattfinden soll das Element „Spieltagsnummer" zwischen 1 und 34 beschränkt werden.

Liveticker

Aus den bei der Ressource Spieltag bereits genannten Gründen werden auch hier aus logischen und funktionalen Gründen redundante Werte wie „Heimmannschaft" und „Gastmannschaft" abgelegt. Diese Werte werden semantisch einem Kommentar zugeordnet. Im Schema sollen sowohl Kommentare als auch Tore optional und ohne obere Grenze implementiert werden. Während eines Fussball-Spiels müssen keine Kommentare bzw. Tore entstehen, theoretisch aber könnten unendlich viele auftreten. Eine Begrenzung der maximalen Anzahl beider Elemente wäre durchaus möglich aber hier nicht sinnvoll.

Mannschaft

Der erste Entwurf des zugehörigen XML-Schemas hatte folgende Elemente:

- Name der Mannschaft
- Kader
- Spieler
- Spielerposition

Allerdings wurde diese Idee während einer Diskussion verworfen. Der Entwurf wäre inkonsistent zu unserer Vorgehensweise die XML-Dateien mit Sicht auf die spätere Implementierung hin zu entwerfen. Aus diesem Grund entwickeln wir eine neue XML-Datei die unserer bisherigen Vorgehensweise entspricht und semantisch besser zu der späteren Umsetzung der Anwendungslogik passt. Es werden für die Funktion „Lieblingsmannschaft festlegen" lediglich die Mannschaften sowie deren Abonnenten gespeichert. In der Bundesliga befinden sich zu jeder Zeit genau 18 Mannschaften. Diese Beschränkung soll ebenfalls im XML-Schema festgelegt werden. Jede der Mannschaften kann von unbegrenzt vielen Personen abonniert werden. Somit besteht auch keine mengenmäßige Beschränkung der abonnierten Mannschaften pro Person.

Tippspiel

Es werden der Name des Tippgebers, sein getipptes Ergebnis und das reale Endergebnis hinterlegt. Somit können die abgegebenen Tipps nach Ende eines Spiels auf Korrektheit überprüft werden und korrekt getippte Ergebnisse können mitsamt des zugehörigen Benutzernamens ausgegeben werden. Auch der eigene Tipp soll überprüft werden können. Alternativ dazu hätte eine Liste mit allen Tipps umgesetzt werden können. Allerdings wären somit die möglichen Funktionen, die auf diese Liste angewendet werden können, stark beschränkt. Die Überprüfung des eigenen Tipps nach Abgabe ist Beispielsweise nicht mehr möglich, da die Werte keiner Person zugeordnet werden können.

4.3 Tatsächlich umgesetzte Ressourcen

Wir haben uns dazu entschieden die Ressourcen "Mannschaft" und "Liveticker" umzusetzen. Die Entscheidung fiel auf diese beiden Ressourcen, da wir mit ihrer Hilfe relativ viele Funktionen abbilden können und sie für asynchrone Funktionen in unserer Applikation ohnehin benötigen.

4.3.1 Mannschaft

Die XML-Datei "Mannschaft.xml" beinhaltet genau 18 Abonnements. Zu Jedem angelegten Abonnement gehört genau eine Mannschaft, die durch ihren Mannschaftsnamen und durch eine eindeutige ID identifiziert werden kann. Diese IDs wurden von uns alphabetisch aufsteigend den Mannschaften zugeteilt. Es muss kein Abonnent für ein Abonnement eingetragen sein, allerdings können theoretisch unendlich viele Benutzer ein Abonnement der selben Mannschaft abschließen. Benutzer können ihre bestehenden Abonnements auch löschen, ohne das es Einfluss auf die anderen Abonnenten einer Mannschaft hat, die der selben Mannschaft folgen. In dem Schema-File "Mannschaft.xsd" sind die Abonnements auf genau 18 festgelegt. Wieso das so ist wurde bereits im oberen Abschnitt erläutert. Eine Restriktion finden wir auch bei Abonnenten wieder, um zu gewährleisten, dass mehrere Abonnenten möglich sind. Die Mannschafts-ID nutzen wir um einen neuen Abonnenten eindeutig in das richtige Abonnement zu schieben und der Mannschaftsname dient uns zur direkten Überprüfung ob der Mannschaftsname auch korrekt ist.

4.3.2 Liveticker

In die XML-Datei "Liveticker.xml" sind zu den ursprünglichen Überlegungen auch noch die Ressourcen "Spiel" und "Tor" geflossen. Hier finden wir 9 Begegnungen zweier Mannschaften wieder, da $18/2 = 9$ ist. Pro Begegnung (Spiel) gibt es jeweils eine Heim- und eine Gastmannschaft, die durch ID und Namen eindeutig erkennbar und zuweisbar sind. Das Element Endergebnis beinhaltet das aktuelle Ergebnis (nach Beendigung eines Spiels das letzte Ergebnis), sowie ein Element Tore, was widerum kein bis unendlich Elemente mit Namen Tor enthält. Pro Tor wird übergeben wer der Torschütze ist, für welche Mannschaft er spielt und in welcher Minute das Tor geschossen wurde. Um den Liveticker komplett zu machen kann man für jedes Spiel Kommentare eingeben, die einen Text enthalten und die aktuelle Spielminute die kommentiert wird. In dem Schema-File ist die erste Restriktion bereits wie erwähnt die Beschränkung der Spielbegegnungen auf 9. Das Element Tor in den Toren ist nach unten und oben unbeschränkt, also kann angefangen bei 0 das Element so oft vor kommen wie man will. Genau das selbe Prinzip findet sich bei Kommentaren wieder. Alle weiteren vorhandenen Elemente sind lediglich mit Namen und Datentyp versehen.

5 Ressourcen und die Semantik der HTTP-Operationen

Uns ist bewusst das PUT mehrfach anwendbar ist und man sowohl etwas neues erzeugen, als auch etwas ändern kann. Nichts desto trotz benutzen wir POST um etwas neu an zu legen und PUT um Daten zu verändern. Zwar wurden nicht alle

Ressourcen die hier auftauchen auch wirklich umgesetzt, jedoch empfanden wir es nicht für unnütz trotz allem die Verben auf alle anfangs geplanten Ressourcen anzuwenden.

Tippspiel

Entstehender Pfad: ../tippspiel/username

GET

Der eigens abgegebene Tipp wird abgerufen. Alle die das richtige Ergebnis vorhergesagt haben werden abgerufen.

PUT

Über die Identifizierung mittels des Namen kann ein abgegebener Tipp geändert werden, solange noch kein Endergebnis des getippten Spiels vorliegt.

POST

Ein neuer Spieltipp wird mit zugehörigem Namen des Nutzers eingegeben.

DELETE

Löschen eines abgegebenen Tipps.

Mannschaft

Entstehender Pfad: ../mannschaft/id=1

GET

Rückgabe einer Fußball-Mannschaft inklusive deren Abonnenten. Als Alternative haben wir folgende Funktion als GET gesehen: Anzeige eines Nutzers inklusive der abonnierten Mannschaften. Wir haben uns bei der Umsetzung gegen diese Methode entschieden, da sie um die Funktionalität zu zeigen nicht notwendig ist.

PUT

Ändern einer Mannschaft nach Auf- bzw. Abstieg in bzw. aus der 1. Bundesliga zu Beginn einer neuen Saison Zu erst haben wir die Funktion wie folgt definiert: Ändern eines Abonnements. Allerdings haben wir uns dazu entschieden, dass man diese Funktionalität auch durch POST und DELETE hat. Deswegen haben wir uns entschieden hier die Mannschaften zu ändern, welche aus der ersten Fußball Bundesliga aufsteigen bzw. absteigen. Am Ende haben wir es ganz verworfen, da wir nur mit einem beliebigen Spieltag arbeiten aus einer einzigen Saison und diese Funktion nicht weiter zu Demonstrationszwecken notwendig ist.

POST

Setzen eines neuen Abonnements Hier brauchen wir keine neue Mannschaft erzeugen lassen, da diese Logik mit PUT erfolgen wird, weil Anzahl der Mannschaften der 1.BL immer bei 18 bleibt.

DELETE

Löschen eines gesetzten Abonnements Das löschen einer Mannschaft macht für uns hier keinen Sinn, da die Mannschaftsanzahl immer bei 18 bleibt.

Liveticker

Entstehender Pfad: ../liveticker/mannschafts_id=1

Während der Überlegungen für die Verben ist uns aufgefallen, dass im XML-Schema ein Element "Ergebnis" fehlt. Somit hatten wir kein sinnvolles Element für das Verb PUT. Sowohl die XML als auch die XSD wurden dahingehend angepasst.

GET

Liveticker eines Spiels wird abgerufen. Das Spiel wird dabei über die Mannschafts-ID identifiziert. Liveticker eines Spieltages wird abgerufen.

PUT

Ändern des momentanen Ergebnisses bzw. des Endergebnisses nach Spielende.

POST

Einen neuen Kommentar zu einem Liveticker hinzufuegen.

DELETE

Entfernen der Kommentare eines Spiels.

Spieltag

Entstehender Pfad: ../spieltag/spieltag_nr=1

GET

Unter Angabe der Spieltags-Nr können die Spielbegegnungen des akutellen bzw. letzten Spieltags abgerufen werden

PUT

(Einzelnes Tor wird über POST hinzugefügt.) Anpassen der Ergebnisse.

POST

Anfangs hatten wir die Idee mittels POST einen neuen Spieltag zu erstellen. Da wir allerdings nur mit einem fiktiven Spieltag arbeiten hat POST folgende Funktion: Hinzufügen eines Tors mit Torschütze, Mannschaft und Minute

DELETE

Spieltag löschen - zu Beginn einer neuen Saison werden die Spieltage der alten Saison gelöscht.

6 RESTful Webservice

Wir haben uns dazu entschieden die Ressourcen Liveticker und Mannschaft umzusetzen, um eine asynchrone Interaktion zu simulieren. Folgende Entscheidungen haben wir bezüglich des Livetickers getroffen: Wir werden die Ticker aller 9 Spiele in einer XML-Datei umsetzen, anstatt für jedes Spiel eine XML zu konstruieren. Das erscheint uns funktional sinnvoller. Der Code wird so auch lesbarer und für uns leichter zu programmieren. Ein Liveticker wird nicht mehr per POST gestartet, sondern besteht aus 9 Spielen, die wir anfangs mit IDs versehen haben. Andere Umsetzungen wären zu komplex. Des Weiteren gibt es ohnehin immer genau 9 Spiele pro Spieltag, die Erzeugung eines Tickers erscheint uns also nicht für sinnvoll. Die IDs haben wir später wieder entfernt, da sie durch eine Umstrukturierung des Codes überflüssig wurden und unser Ziel es unter Anderem ist überflüssige Dinge, besonders IDs, zu vermeiden. Folgende Entscheidungen haben wir bezüglich der Mannschaften getroffen: Anders als beim Liveticker haben wir bei den Mannschaften IDs. Wir halten sie für sinnvoll um die gesuchte Mannschaft zu identifizieren. Die logische Alternative dazu wäre die IDs zu entfernen, wobei man dann Strings vergleichen müsste und jeder Tippfehler zu einem Fehler führen würde. Da wir davon ausgehen, dass so ein Eingabefehler bei den Mannschaftsnamen schnell mal passieren kann sind wir bei den IDs geblieben. Anfangs war es für einen Abonnenten nur möglich genau ein Abonnement zu aktivieren, also lediglich einer Mannschaft zu folgen. Auch das haben wir überarbeitet, da wir diese Beschränkung nicht für sinnvoll hielten.

7 Konzeption asynchrone Kommunikation

7.1 Mannschaft

7.1.1 Leafs (Topics)

Als Leafs (Topics) existieren hier die Mannschaften.

7.1.2 Publisher/Subscriber

Der Subscriber ist die Person die einer bestimmten Mannschaft folgt. Über die Ressource “Liveticker” werden dann die zugehörigen Informationen publiziert.

7.1.3 Übertragene Daten

Abonnenten erhalten eine Benachrichtigung sobald ein Tor oder ein Kommentar in einem Spiel der abonnierten Mannschaft hinzugefügt wird. Da diese Elemente an ein Spiel gekoppelt sind werden auch Benachrichtigungen publiziert wenn eine der genannten Aktionen bei der gegnerischen Mannschaft auftritt.

7.2 Liveticker

7.2.1 Leafs (Topics)

Über die Mannschafts-ID wird der Ticker des Spiels abgefragt, an dem die über die ID identifizierte Mannschaft teilnimmt.

7.2.2 Publisher/Subscriber

Der Abonnent wird benachrichtigt sobald der Server neue Daten des ausgewählten Spiels erhält. Da die Spieldaten von einer Person eingepflegt werden ist der Client auch Publisher, da dieser dem Server die benötigten Informationen mitteilen muss. (Zwei Konsolen benötigt - Einmal zum "publishen" und einmal zum "subscriben")

7.2.3 Übertragene Daten

Je nach Ereignis-Eintritt wird dem Subscriber das Element "Tor" bzw. "Kommentar" übermittelt. Diese Daten werden zuvor von einem Client (Admin) auf den Server übertragen. Als Daten werden bei einem Tor Werte für Torschütze, die Mannschaft des Torschützen sowie die Spielminute übermittelt. Ein Kommentar enthält einen Text und die zugehörige Spielminute.

Weitere Gedankengänge und Entscheidungen

An dieser Stelle kam uns die Idee jedem Benutzer zu ermöglichen eigene Kommentare zu schreiben während ein Liveticker aktiviert ist. Diese Idee wurde auf Grund von mangelnder Zeit und Abgabefristen verworfen.

8 Einrichten des XMPP-Servers

Wir haben den Server (Openfire) auf Mac OS X installiert. Hierbei gab es ein Problem die uns einiges an Zeit gekostet hat. Bei dem auftretendem Problem konnten wir uns nicht als Admin anmelden. Das kam dadurch, dass bei der Registrierung ein falsches Zeichen beim Passwort übergeben wurde, weil wir nicht mit dem Cursor auf "weiter" geklickt, sondern mit "ENTER" bestätigt haben. Das Problem konnte nach einem Gespräch mit einem der Betreuer und dessen Hilfe behoben werden. Das nächste Problem vor dem wir standen was die fehlerhafte SMACK API. Angefangen zu arbeiten haben wir mit der Version 3.2.2. Nach diversen Problemen und Versuchen des Bugfixing konnten alle Probleme durch ein Update der API auf Version 3.3.0 behoben werden.

9 XMPP-Client

Damit der Client funktionsfähig ist müssen als erstes die benötigten Knoten vorhanden sein. Um dies zu gewährleisten erstellen wir mit der Klasse “CreateNodes.java” die 18 Knoten unserer 18 Mannschaften. Die erstellten Knoten sind persistent, damit sie nach einen Neustart des Servers noch vorhanden sind. Des Weiteren beschreiben wir an dieser Stelle mit Hilfe von XMPPConnection den Log-in Vorgang. Bei der Entwicklung des Clients entstanden erst die Klassen “PubClient” und “SubClient”. Um unnötige Redundanzen zu vermeiden und zur besseren Übersichtlichkeit wurden beide Klassen zu einer Klasse “PubSubClient” zusammengefasst. In dieser Klasse sind alle Funktionen auf die wir zurückgreifen wollen hinterlegt. Was genau welche Funktion macht haben wir als Kommentar über die jeweilige Funktion geschrieben. Nichts desto trotz werden wir im Folgenden auf ein paar Auszüge unseres Codes eingehen.
Als erstes kommt die Verbindung zum Server mit passenden Log-In Informationen (Nutzername und Passwort) die übergeben werden.

```
public PubSubClient(String user, String pass) throws XMPPException {
    connection.connect();
    connection.login(user, pass);
}
```

Nach einigen Funktionsabläufen wird ein Disconnect durchgeführt um Fehler durch mehrfache Verbindungsversuche zu vermeiden. Dies ist notwendig, da die Verbindung immer durch den Konstruktor hergestellt wird.

```
//Verbindung trennen
public void disconnect() {
    connection.disconnect();
}
```

Um den Payload kompatibel mit den erstellen XML-Schemata zu versenden wurde hier die bestehende XML-Struktur nachgebildet. Somit ist die Kompatibilität zu den XML-Dateien gegeben.

```
//Publiziert Kommentar
public void pubComment(String team, int min, String comment) throws XMPPException {
    LeafNode node = mgr.getNode(team);
    SimplePayload payload = new SimplePayload("Livticker", null, "<Spiel><Kommentare><Kommentar><Minute>" + min + "</Minute><Text>" + comment + "</Text></Kommentar></Kommentare></Spiel>");
    PayloadItem item = new PayloadItem<SimplePayload>(team, payload);
    node.publish(item);
    System.out.println("Kommentar erzeugt.");
}
```

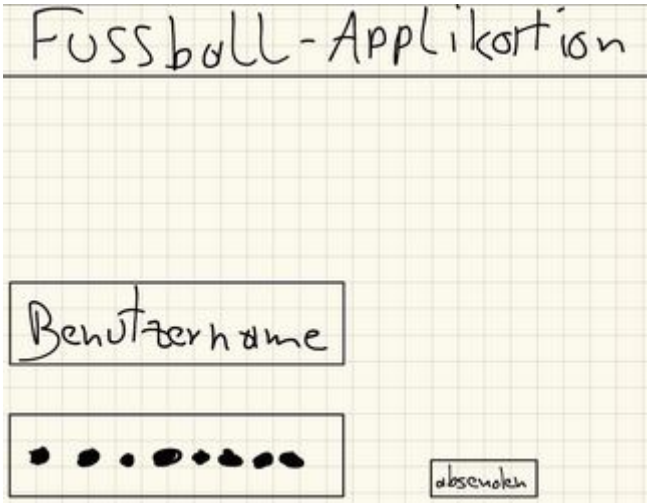
Die Funktion der letzten Methode ist zum löschen aller Knoten. Die haben wir für den Fall entwickelt, falls das Erzeugen aller Knoten getestet werden soll oder etwas schief gegangen ist was nicht rückgängig zu machen ist. In diesem Fall kann man so leicht die Knoten löschen und neu erstellen.

10 Graphical User Interface

10.1 Entwurf

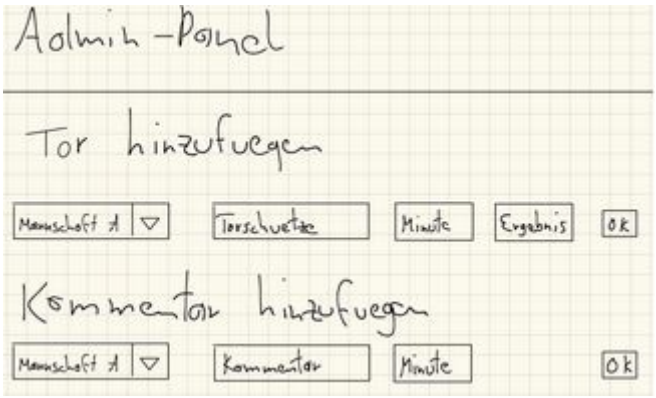
Während der Entwurfsphase kamen uns folgende Gedankengänge:
Unser Log-In-Fenster wird mittig vom Screen ausgegeben. Die gesetzte Überschrift “Fussball-Applikation” ist fest verankert. In unserem Entwurf für das

Log-In Fenster befinden sich zwei Eingabefelder. Ein Feld zur Eingabe des Benutzernamens und eins für das entsprechende Passwort zum Benutzer. Hier setzen wir auf Selbstbeschreibbarkeit, indem wir die Worte “Benutzername” und “Passwort” in den Feldern vorgeben. Diese sollen verschwinden sobald man in das entsprechende Feld klickt. Zum Bestätigen des Log-Ins existiert ein Submit-Button mit der Beschriftung “ansenden”. Wenn der Log-In nicht erfolgreich durchgeführt werden konnte wird das ganze Formular, also die beiden Textfelder, wieder zurückgesetzt und es erscheint ein Pop-Up mit dem Hinweis das etwas schief gelaufen ist. Falls der Log-In erfolgreich war schließt sich das Fenster und wir kommen zur nächsten GUI. Hierbei haben wir vor zwischen der Administrator und der Benutzersicht zu unterscheiden.



Administratorkonsole

Falls man sich als Administrator eingeloggt hat, kann man eine Mannschaft per Dropdown Menü wählen und ein gewünschtes Ereignis hinzufügen (Tor oder Kommentar im Liveticker). Wir entscheiden uns bei der Auswahl der Mannschaft für ein Dropdown Menü, da wir als einzig sinnvolle Alternative Checkboxen sehen, die allerdings hier nicht nützlich sind, da wir immer nur auf eine Mannschaft gleichzeitig zugreifen können und sich Checkboxen aus unserer Sicht nur bei einer Mehrfachauswahl lohnen. Des Weiteren wollen wir die GUI so strukturiert und kompakt wie möglich konstruieren, was im Falle von 18 Checkboxen nicht der Fall wäre und zur totalen Überlagerung kommen würde.



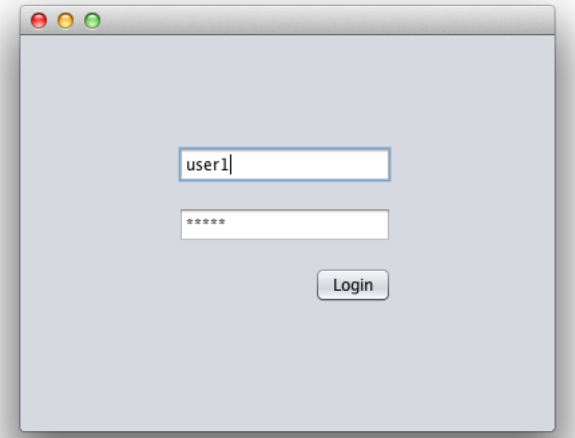
Benutzerkonsole

Das User-Interface sieht im großen und ganzen dem Admin-Panel sehr ähnlich. Klein, kompakt und leicht minimalistisch soll es werden. Auch hier haben wir uns für ein Dropdown Menü entschieden wenn es um die Mannschaftswahl geht um das Design konsistent zu halten und die GUI nicht zu überlagern. Es gibt hier ein zweites Dropdown Menü, was dazu dient, eine gewünschte Funktion auszuführen (z.B. Abonnieren einer Mannschaft). Im Ausgabefeld wird dann das Ergebnis des Funktionsaufrufs ausgegeben. Wir hatten für beide GUIs Alternativen die sich im Anhang wiederfinden. Unser Ziel ist die automatische Erkennung der Rechte (User/Admin) während des Log-Ins und somit das starten der richtigen GUI zum richtigen Log-In. Wir wissen momentan noch nicht wie diese Idee umzusetzen ist, deswegen haben wir sie nach hinten verschoben und als "optional" auf unsere ToDo-Liste gesetzt.

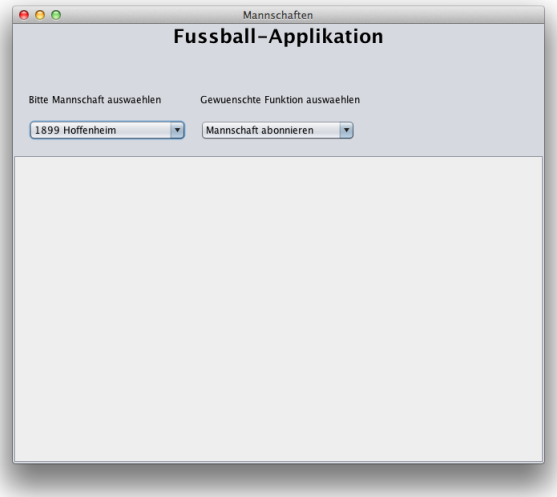
10.2 Entwicklung

Den ersten Versuch eine GUI mit Swing zu erstellen haben wir in der Entwicklungsumgebung "IntelliJ IDEA" unternommen. Bei Swing stießen wir auf Probleme beim Verständnis mit dem Umgang des Interfaces. Die Instabilität einiger Plugins (sowohl für IntelliJ IDEA als auch Eclipse) haben uns schließlich

dazu bewegt andere Wege zu gehen. Der nächste Versuch wurde mit Hilfe von NetBeans unternommen. Da beide Mitglieder unseres Teams mit IntelliJ IDEA arbeiten und bisher keine Erfahrungen mit NetBeans gemacht haben wurde auch dieser Versuch abgebrochen. Nachdem wir uns kurzzeitig dazu entschieden haben die GUI in "Handarbeit" zu erstellen gingen wir aus Gründen der Komplexität doch wieder zurück zu NetBeans. Den von NetBeans generierten Code für die Oberfläche haben wir in IntelliJ IDEA importiert und die notwendigen Action Listener dort geschrieben, da die benötigten Methoden (XMPP Client) auch dort lagen. Alle Variablen für die Fenster wurden von uns umbenannt. So wurde aus "jlabel1" bspw. "jlabeltitle". Das geschah um die Semantik im Code zu verbessen. Das Programm startet immer mit dem Log-In Fenster.



Sobald man es geschafft hat sich erfolgreich einzuloggen kommt man in das Main Menü was auf dem folgenden Screenshot zu sehen ist.



Dieses Fenster ist ein Zusammenschluss von unserem geplanten Admin-Panel

und dem User-Client. Da bei der geringen Masse an Funktionen keine zwei seperaten GUIs notwendig waren haben wir uns für diesen Weg entschieden und alle Funktionen in eine gemeinsame GUI fließen lassen. Beim connect erstellen wir einen neuen PubSub-Client der sich automatisch verbindet. Dieser Client wird von uns von Fenster zu Fenster übergeben (Main Window / Admin Window) damit wir weiterhin mit dem Client arbeiten können wenn wir uns einmal eingeloggt haben. Wenn man sich alle Mannschaften anzeigen lassen will erscheint die Ausgabe bei uns leider noch in der Konsole und nicht in dem dafür vorgesehenem Feld unter dem Auswahlmenü. Wenn man eine Mannschaft per Dropdown-Menü gewählt hat und die Funktion auswählt auf die man zugreifen möchte wird diese ohne weiter notwendige Bestätigung durchgeführt. Für die Funktionen "Tor senden" und "Kommentar veröffentlichen" öffnet sich das "Publishing"-Fenster, auf dem beide Funktionen möglich sind.

Die Auswahl der Mannschaft wird bei jeder Funktionswahl in einer Variablen gespeichert und kann so von jedem Fenster abgerufen werden, ohne sie an irgendeiner Stelle erneut wählen zu müssen. Zum Abschluss noch ein paar Worte zur Logik unseres Dropdown-Menüs. Wir haben mit einer einfachen switch-case Funktion gearbeitet, die je nachdem welcher Menüpunkt gewählt ist diesen ausführt und rausspring durch das "break", wie es sein sollte. Beim Fall 4 und 5 (Tor senden und Kommentar verfassen) wird das selbe Fenster aufgerufen, deswegen verweist Fall 4 auch auf Fall 5.

```
switch (function) {
    case 0:
        pubSub.subscribe(team);
        JOptionPane.showMessageDialog(null, "abonniert");
        break;
    case 1:
        pubSub.unsubscribe(team);
        JOptionPane.showMessageDialog(null, "Abonnement beendet");
        break;
    case 2:
        pubSub.discover();
        break;
    case 3:
        pubSub.getMessagesFromNode(team);

        break;
    case 4:
    case 5:
        dispose();
        AdminWindow adminWindow = new AdminWindow();
        adminWindow.setPubSub(pubSub);
        adminWindow.setTeam(team);
        adminWindow.setVisible(true);
}
```

Über unsere GUI wird bei der Auswahl einer Funktion sowohl die XMPP- als auch die RESTbasierte Methode aufgerufen (außer es wäre sinnlos). Für eine höhere Gebrauchstauglichkeit gibt es Bestätigungspopups wenn eine Funktion erfolgreich ausgeführt wurde. Die Ergebnisse (oder Rückgabewerte) werden als XML-Struktur unformatiert in der GUI ausgegeben, somit wird direkt auf die REST-Struktur bzw. den XMPP-Payload zurückgegriffen.
Bei der asynchronen Kommunikation sind wir soweit gekommen das ein Pop-Up Fenster geöffnet wird falls eine neue Nachricht auftritt. Schlussendlich hat auch die Ausgabe in der GUI funktioniert, die ja eine Weile nur in der Konsole möglich war.

11 Fazit

Die Entwicklung der webbasierten Anwendung, in unserem Fall eine Fussball-Applikation, wurde abgeschlossen. Die in der Vorlesung vorgestellten Services und Architekturen konnten in der Praxis erfolgreich eingesetzt bzw. implementiert werden. Während der Entwicklung konnten wir verschiedene Erkenntnisse gewinnen, die durch eine rein theoretische Vermittlung nicht erkennbar gewesen wären. Dazu gehören verschiedene Entscheidungen und Probleme, die während der Entwicklung auftreten und in diesem Dokument festgehalten sind. Durch diesen Entwicklungsweg konnten ebenfalls viele neue Kenntnisse gewonnen werden. Zusammenfassend ergibt der Workshop für uns eine sinnvolle Ergänzung und Vertiefung zu den theoretisch vermittelten Kenntnissen aus der Vorlesung des Moduls "Webbasierte Anwendungen 2".