MASTERARBEIT SOFTWAREENTWICKLUNG - WIRTSCHAFT

SIMON ANTONIO TOMÁS JIMÉNEZ, BSC 0530901

CUSTOMIZABLE WEB INTERFACE WITH DYNAMIC TEMPLATE GENERATION
DEVICE INDEPENDENT DYNAMIC DATA DISPLAY

ANPASSBARES WEB INTERFACE MIT DYNAMISCHER TEMPLATE-GENERIERUNG
GERÄTEUNABHÄNGIGE DYNAMISCHE DATENANZEIGE

Magisterarbeit vorgelegt zur Erlangung des akademischen Grades eines Diplomingenieurs der Studienrichtung Softwareentwicklung-Wirtschaft

Betreuer:

AO.UNIV.-PROF. DIPL.-ING. DR.TECHN. NIKOLAI SCERBAKOV
Institut für Informationssysteme und Computer Medien, TU Graz

Graz, 04.03.2013

DANKSAGUNG

Ich möchte mich bei meinem Betreuer Dr. Scerbakov bedanken, der mir das Erforschen dieses Themas ermöglicht hat und mir vertraut hat, hier eine eigenständige Arbeit zu verfassen.

Vielen Dank für die Unterstützung geht an Mario, Marc und Manuel von Global Sport Services, die mir mit ihrem Fachwissen weitergeholfen haben. Ohne euch wäre diese Arbeit nie das geworden, was sie jetzt ist.

Der größte Dank gilt natürlich meiner Familie, meinen Eltern für die kompromisslose Unterstützung und meiner Freundin fürs Korrekturlesen und die Geduld, wenn meine Zeit für sie knapp geworden ist.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 04.03.2013 Simon Antonio Tomás Jiménez

KURZFASSUNG

Bei der Suche nach einem Thema, bei dem eine geräteunabhängige Darstellung von dynamischen Daten Sinn macht, gibt es viele Möglichkeiten. Schlussendlich wurde das Thema "Skirennen" aufgegriffen. Als unkonventionelles Thema bietet es viele spannende Herausforderungen, die jeweils einer neuen Überlegung bedürfen.

In dieser Masterarbeit soll ein möglichst intelligenter Mix zwischen fest einprogrammierten und dynamischen Teilen dargestellt werden, der eine maximale Performance der Applikation bei größtmöglicher Benutzbarkeit und Flexibilität erlaubt.

In diesem Projekt sollen also die (zufallsgenerierten) Live-Daten eines Skirennens als Beispiel genommen werden. Diese sollen, um die höchstmöglichste Performance zu bieten, auch intelligent zwischengespeichert werden, was eine weitere Herausforderung darstellt. Diese Daten sollen für vier verschiedene Anzeigeformen aufbereitet werden:

- Live im Internet
- Live im lokalen Wireless LAN
- Live über ein Fernseh-Signal
- Live auf Mobilen Geräten.

Diese Anzeigeformen müssen als Annahme nicht nur für die momentan bekannten sechs verschiedenen Formen von Skirennen funktionieren, sondern auch für die verschiedenen Zielpersonen, wie z.B. Fernsehsprecher, Trainer, Rennbegleiter, Zuseher.

Aufgrund der verschiedenen Formen dieser Skirennen ist eine fixe Einteilung der Daten sowie Screens nicht möglich, die möglichen Kombinationen sind nicht abschätzbar. So soll eine Verwaltung geschaffen werden, mit deren Hilfe die verschiedenen Anzeigen, teilweise während der Laufzeit, angepasst werden können. Das inkludiert:

- Die Anzahl, Reihenfolge, Bezeichnung und Aktualisierungsart der verschiedenen Spalten der angezeigten Datenreihen
- Die Position, Größe und Art von Anzeigefenstern

Diese Arbeit basiert auf einem gegebenen Datenbank-Layout, in die von einem Zeitnehmer der FIS [1], vom Internationalen Skiverband, die Ergebnisse der Rennen eingetragen werden.

Als Hauptsprache wurde PHP gewählt. Die Oberflächenverwaltung soll mit einer Kombination aus HTML, JavaScript, CSS und PHP / mySQL erstellt werden. Funktionsfähig soll diese in aktuellen Versionen von Chrome (v23) [2], Firefox (v17) [3], Opera (v12) [4] und Internet Explorer sein. Die Anzeige soll in Chrome, Firefox, Opera, Internet Explorer [5] und gängigen mobilen Geräten (Android (v2 & v4), iOS(v5, v6)) funktional sein.

Als technologische Themenweiterführung der Bachelorarbeit "Sichere und dynamische Navigation in einer zufallsgenerierten Umgebung" und des Masterprojekts "Modulares Chat-System, integrierbar in existierenden E-Learning Systemen" soll diese Arbeit ausschließlich die Teile erklären, die ein höheres Verständnis erfordern. Alle Techniken und Überlegungen, die in diesen Arbeiten beschrieben sind, werden in dieser Arbeit ausgelassen.

ABSTRACT

There are many topics in the field of "Device independent dynamic data display with live template generation". To get some challenges in this field, the topic "Ski Races" was chosen. The goal of this master thesis is to create an intelligent mix between hardcoded and dynamic parts, which allows high usability and flexibility without a big impact on performance.

The (randomly generated) live-data of Ski races shall be used in this project. To achieve high performance, this data should be cached in an intelligent way, which creates another challenge. This data shall be prepared for four different displays:

- Live on the Internet
- Live on local Wireless LAN
- Live over a TV set
- Live on Mobile Devices

We assume that these display forms must work for the given six different ski race types and for the very specific needs of the viewer, e.g. the television announcer, the trainer, the visitor.

Because of the different forms of ski races, the display method cannot be predefined. Therefore a management-system should be created that can adapt the display form to these specific needs during the usage, including:

- The amount, order, description and method of actualization of the data sets
- The position, size and type of the modular windows

This work bases on a given Database-Layout, in which the timekeeper inserts the results of the ongoing ski events organized by FIS, the international ski federation.

The main programming language is PHP. The templating system shall be realized with a mix of HTML, JavaScript, CSS, PHP / mySQL. The goal is to have the templating system working in latest Versions of Chrome (v23), Firefox (v17), Opera (v12) and Internet Explorer (v9). The data-display shall

work in latest Versions of Chrome, Firefox, Opera, Internet explorer and usual mobile Browsers (Android, iOS).

As a technology-based topic continuation of the Bachelor Thesis "Dynamic and Secure Navigation in a randomly generated Environment" and the Master Project "Modular chat-system, integrate-able into existing E-Learning Systems", this work shall only describe the parts that require a better understanding of the topic. Techniques and ideas described previously will be left out.

INHALTSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

Bracket .....

Wenn ein Finale aus vielen Heats besteht, werden diese in Brackets zusammengefasst. Teilweise mit KO-System, mit 2 oder 4 Teilnehmern.

CIS ..... *Siehe* CIS

Competitor .....*Ein Teilnehmer an einem Wettbewerb*

Heat .....*Ein Teilbewerb eines Wettbewerbs*

Judges.....

..... Punkte oder sonstige Richter. Meist bestimmen mehrere davon das Rennergebnis.

Run..... *Ein Durchgang eines Teilbewerbs oder Wettbewerbs; Auch: Ein Teilbewerb*

Tie .....*Punkte oder Zeitgleichheit zwischen zwei Teilnehmern*

ABBILDUNGSVERZEICHNIS

CODEBEISPIELE

AUFGABENSTELLUNG - MOTIVATION UND ZIELE

MOTIVATION

GERÄTEUNABHÄNGIG

Die Verwendung von mobilen Geräten zum Browsen im Internet nimmt seit 2010 stark zu. Haben zu 2010 laut StatCounter nur 3.81% der Benutzer einen mobilen Browser benutzt [6], waren es Ende 2012 bereits 14.55% [7]. Natürlich sind diese Zahlen basierend auf den Seiten erstellt, die diese proprietäre Zählungsmechanismen verwenden. Andere Quellen geben für unterschiedliche Länder verschiedene Daten an [8]:

Mobile share of Web Traffic all across the World

Region	2010	2012
Africa	5.81%	14.85%
Asia	6.1%	17.84%
Europe	1.81%	5.13%
North America	4.71%	7.96%
South America	2.88%	7.55%
Oceania	1.46%	2.86%
World Wide	3.81%	10.01%

Abbildung 1: Mobile Market Share, Go-Globe.com

Hier zeigt sich, dass z.B. in Europa der Anteil an mobilen Zugriffen mit 5,13% noch eher gering ist, vor allem verglichen mit Extrembeispielen wie Zimbabwe (58,06%) oder Indien (48,24%). Natürlich erklärt sich der hohe Anteil an mobilen Zugriffen auch dadurch, dass in einigen Ländern der Ausbau vom Festnetz nicht kosteneffektiv ist bzw. durch die Investitionshöhe einfach noch nicht durchgeführt werden konnte. Als plakatives Beispiel kann man Österreich nehmen, in dem 79% der Haushalte einen Internetzugang haben, 77% sogar eine Breitbandverbindung (> 144 kBit/s). 98% aller österreichischen Unternehmen sind am Internet angeschlossen [9].

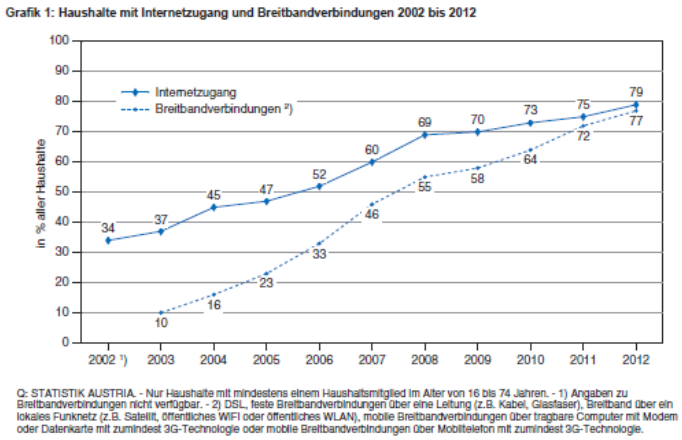


Abbildung 2: Entwicklung der Internetanschlüsse seit 2002, Statistik Austria

Diese allgemeinen Zahlen können als Hinweis dienen, wie der echte Traffic auf der Seite aussehen wird. Glücklicherweise gibt es solche Seiten bereits. Bei der Erstellung der Seite kann also auf echte Daten zugegriffen werden.

Wie sieht die Verteilung also "hier" aus? Eins steht fest: Sehr variabel. Von Event zu Event und Austragungsort zu Austragungsort (die dem Internationalen Charakter zufolge klarerweise sehr divergierend sind) gibt es gravierende Unterschiede. Als Beispiel wurde der Ski-Cross Event in "Deer Valley" gewählt.

Mit welchen Browsern wurde auf den Event zugegriffen? Nachdem 28% der Zugriffe von mobilen Geräten stammten, war die Richtung bereits klar. Doch auch hier gibt es noch feine Unterschiede - so ist auf IOS-Geräten wie iPhone oder iPad Safari der vorinstallierte Browser, auf Android-Geräten ein anderer WebKit-basierter Browser [10].

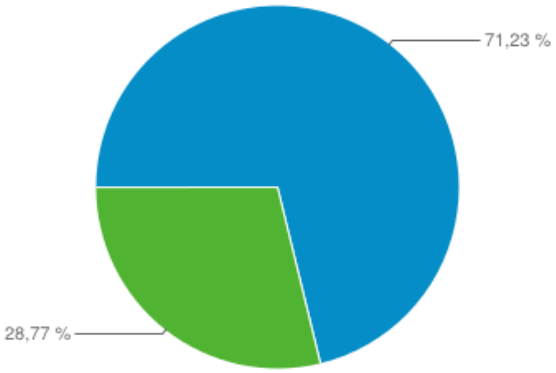


Abbildung 3: 28.77% der Zugriffe bei diesem Event waren von Mobilen Geräten

Von den meistgenutzten 5 Browsern benutzen 4 (Chrome, Safari, Safari In-App, Android Browser) die WebKit-Rendering-Engine [11] [12] [13] [14]. Chrome und Safari unterscheiden sich allerdings im verwendeten JavaScript-Core, und auch der vorinstallierte Android-Browser hat ein anderes Verhalten als die anderen drei Browser. So kann man davon ausgehen, dass die meistverwendeten sechs Browser alle ein verschiedenes Verhalten haben werden. [15]

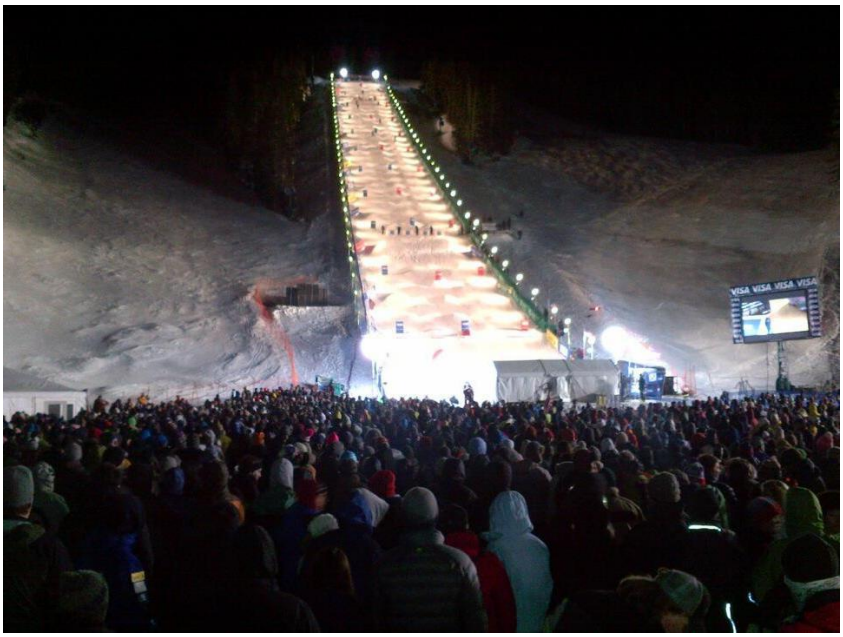


Abbildung 4: Ski Cross, Foto: fisfreestyle.com

Browser	Besuche	Besuche	Anteil am Gesamtergebnis: Besuche
1. Chrome	2.673	28,97 %	
2. Safari	2.468	26,75 %	
3. Firefox	1.733	18,78 %	
4. Safari (in-app)	808	8,76 %	
5. Android Browser	743	8,05 %	
6. Internet Explorer	699	7,58 %	
7. Opera	74	0,80 %	
8. Mozilla Compatible Agent	11	0,12 %	
9. Lunascape	3	0,03 %	
10. Opera Mini	3	0,03 %	
11. BlackBerry9700	2	0,02 %	
12. Playstation 3	2	0,02 %	
13. RockMelt	2	0,02 %	
14. SAMSUNG-GT-S7070	2	0,02 %	
15. BlackBerry9300	1	0,01 %	
16. Camino	1	0,01 %	
17. IE with Chrome Frame	1	0,01 %	
18. SeaMonkey	1	0,01 %	

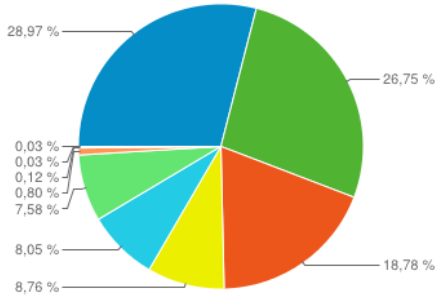


Abbildung 5: Browseraufteilung beim existierenden Live - Scoring Tool

Die verwendeten Geräte (und damit auch die verfügbare Bildschirmgröße) variieren auch stark. Die meisten Besucher sehen die Seite mit einem iPhone, danach in absteigender Reihenfolge iPad, Samsung S II, Samsung S3, Samsung Note II und andere. Die verwende Bildschirmauflösung ist mit 17,34% am häufigsten 1355x768, dicht gefolgt von 1280x800 (15,89%). Auf dem dritten Platz befindet sich hier allerdings 320x480 (9,02%), am vierten 1920x1080 (6,55%).

1.	1366x768	323	17,34 %
2.	1280x800	296	15,89 %
3.	320x480	168	9,02 %
4.	1920x1080	122	6,55 %
5.	1440x900	120	6,44 %
6.	1600x900	111	5,96 %
7.	768x1024	101	5,42 %
8.	1280x1024	90	4,83 %
9.	1680x1050	89	4,78 %
10.	320x568	53	2,84 %

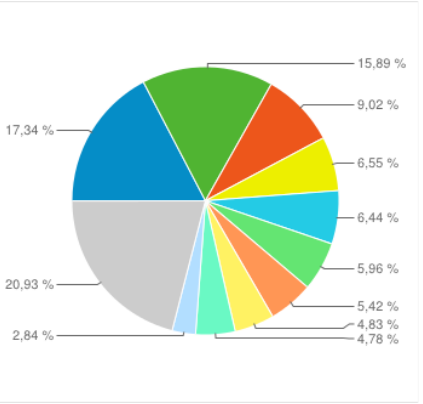


Abbildung 6: Verwendete Auflösungen

Java-Unterstützung bieten 55,72% der Geräte, die Flash-Unterstützung lässt sich fast vom Betriebssystem auslesen: 27% der Geräte (Desktop sowie Mobil) unterstützen Flash nicht; iOS hat 17% Anteil an der gesamten Statistik.

1.	Windows	986	52,93 %
2.	Macintosh	333	17,87 %
3.	iOS	316	16,96 %
4.	Android	200	10,74 %
5.	SymbianOS	10	0,54 %
6.	Windows Phone	7	0,38 %
7.	Linux	5	0,27 %
8.	(not set)	3	0,16 %
9.	BlackBerry	3	0,16 %

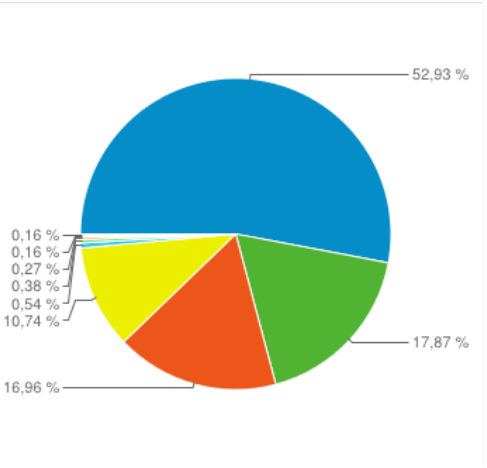


Abbildung 7: Verwendete Betriebssysteme

Aus diesen Statistiken ergibt sich sehr eindeutig: Eine einzelne, vorgefertigte Lösung wird hier nichts bringen. Die Anzeige kann auch nicht in einer gewissen Anzahl an vorgefertigten Varianten einprogrammiert werden - hier muss es Benutzereinstellungsmöglichkeiten geben, mit Hilfen um die verschiedenen Bedürfnisse zu befrieden.

Die fertige Lösung soll also unabhängig vom verwendeten Gerät und Betriebssystem, der verwendeten Bildschirmgröße, Auflösung, Flash und Javaunterstützung funktionieren. Da gerade im Anwendungsfall "auf der Piste" mit einer langsamen Verbindung gerechnet werden muss, darf das System auch nicht zu viel Traffic verursachen.

Die Alternativen ließen sich schon stark eingrenzen. Das einzige, was "überall" funktioniert, ist HTML - und Javascript. Javascript wird laut Web Aim von 98.6% der Internet-Benutzer verwendet [16]. Von diesen 1.4% haben 70% Firefox benützt (was Rückschlüsse auf das abschaltbare NoScript-Plugin erlaubt) und 17% den textbasierten Browser Lynx. Ausgehend von dieser Annahme kann man behaupten, das 99% der Internet-Surfer JavaScript unterstützen oder zumindest unterstützen können wenn sie wollen. Javascript ist auf aktuellen Mobilen Geräten (Handys, Tablets) um den Faktor 4-10x langsamer als auf aktuellen PCs (Laptops, Standrechner) [17] [18]. Da die Ausführungsgeschwindigkeit allerdings generell nicht besonders hoch eingeschätzt wird, sollte es auch ohne große Probleme auf langsamen mobilen Geräten funktionieren.

Es bleiben 2 Alternativen: eine statische Seite, die die Ergebnisse anzeigt, und eine dynamische Seite, die die Ergebnisse nachlädt. Aufgrund der geringen Penetrationsrate von Benutzern ohne JavaScript (und da es sich kaum auszahlt, über eine statische Seite eine Diplomarbeit zu schreiben), fällt die Entscheidung an dieser Stelle für ein gemischtes System aus einer dynamischen Seite mit einem Backend.

Diese Entscheidung führt zu zwei weiteren Fragen: Welches System soll die dynamische Seite rendern, welches System soll das Backend benutzen?

Die dynamische Seite kann alle möglichen Daten empfangen - z.B. Text, Javascript-Code, XML oder JSON. Das verwendete Objekt XMLHttpRequest (die Implementation der ECMAScript HTTP API [19]) ermöglicht die Übertragung von allen möglichen Daten, auch wenn der Name aus historischen Gründen hier in eine etwas falsche Richtung deutet [20].

THEORETISCHE GRUNDLAGEN

CACHING VON DYNAMISCHEN DATEN

Das Caching stand bereits am Anfang der Software, auch wenn zu diesem Zeitpunkt weder der Umfang noch die genauen Anforderungen des Projekts klar waren. Im Hinblick auf die "Effects of Change", die in Rapid Development von Steve McConnell [21] beschrieben werden, wurde dieses Modul absichtlich vorgezogen.
Damit konnte ein späteres Umschreiben von vielen Modulen umgangen werden. [1]

Um ein effektives Caching zu ermöglichen, muss die Abfrage, ob es ein gecachtes Ergebnis gibt, sofort oder zumindest extrem schnell erfolgen. Als erste Möglichkeit wurde ein flat-file caching [22] angedacht. Das bedeutet, dass die Ergebnisse in eine Art vorgenerierte Textdatei geschrieben werden, und nur wenn diese Datei nicht gefunden wird, wird diese Datei generiert [23]. Diese Möglichkeit musste verworfen werden, da äußerst unklar ist, zu welchem Zeitpunkt neue Daten in das System kommen. Eine weitere Lösung wäre ein reiner mySQL- Querycache gewesen, z.B. über einen View [24] oder Stored Procedures [25].

Die endgültige Lösung wurde ein Mix: Eine Abfrage, ob eine vorgenerierte Datei da ist, abgefangen über den automatischen Query Cache [26] von MySQL, zusammen mit einem System, dass diese Dateien inkrementell vorgeneriert. Die genaue Umsetzung wird im praktischen Teil beschrieben.

CACHING VON STATISCHEN DATEN

Jede Website besteht aus mehreren Dokumenten, von denen sich einige nur unregelmäßig ändern. Beispielsweise ist es meistens nicht nötig, eine CSS - Datei für jede HTML-Seite neu zu laden, da der Inhalt dieser Datei sich nicht ändert.

Es gibt in HTTP/1.1 "starke" und "schwache" Header, die das Neu-laden der Ressource beeinflussen [27]. Starke Header sind z.B. `Expires` und `Cache-Control: mag-age`. Damit wird spezifiziert, wie lang eine Ressource gültig bleibt ohne neu geladen werden zu müssen. In dieser Zeit wird vom Client keine Anfrage an den Server betreffend

[1] Die allgemein benutzte Vorgehensweise, Evolutionary Delivery, ist ebenso in Rapid Development [21] beschrieben, S. 425ff.

dieser Datei geschickt, was die Ladezeit auf die Geschwindigkeit des lokalen Caches reduziert.

`Last-Modified` und `ETag` spezifizieren Charakteristiken über die Ressource, die der Browser überprüft. In `Last-Modified` steht (am besten…) das Datum, an dem die Ressource das letzte Mal geändert wurde.
Z.B. schickt der Client bei der zweiten Anfrage dieses Datum mit - der Server kann dann auf den Request mit `304 - Not Modified` antworten [28]. Der `ETag` kann den Inhalt der Datei vorberechnen und diesen als Hash-Wert übermitteln.
Die Technik, die am ehesten garantiert, dass die Dateien so selten wie möglich aber garantiert geladen werden wenn sie sich geändert haben, nennt sich Fingerprinting. Dabei wird der Ressource ein Fingerabdruck verpasst, der dafür sorgt, dass sie garantiert neu geladen wird, z.B. eine Zahl vor oder nach dem Dateinamen. Das wäre z.B. `/static/javascript/v14/include.js` statt `/static/javascript/include.js`. Serverseitig kann die Anfrage auf die richtige Datei geleitet werden, z.B. mit einer `mod_rewrite` Anweisung. Bei meinem Masterprojekt wurde Fingerprinting angewandt.

Ein Best-Practice Beispiel liefert wie so oft Google:

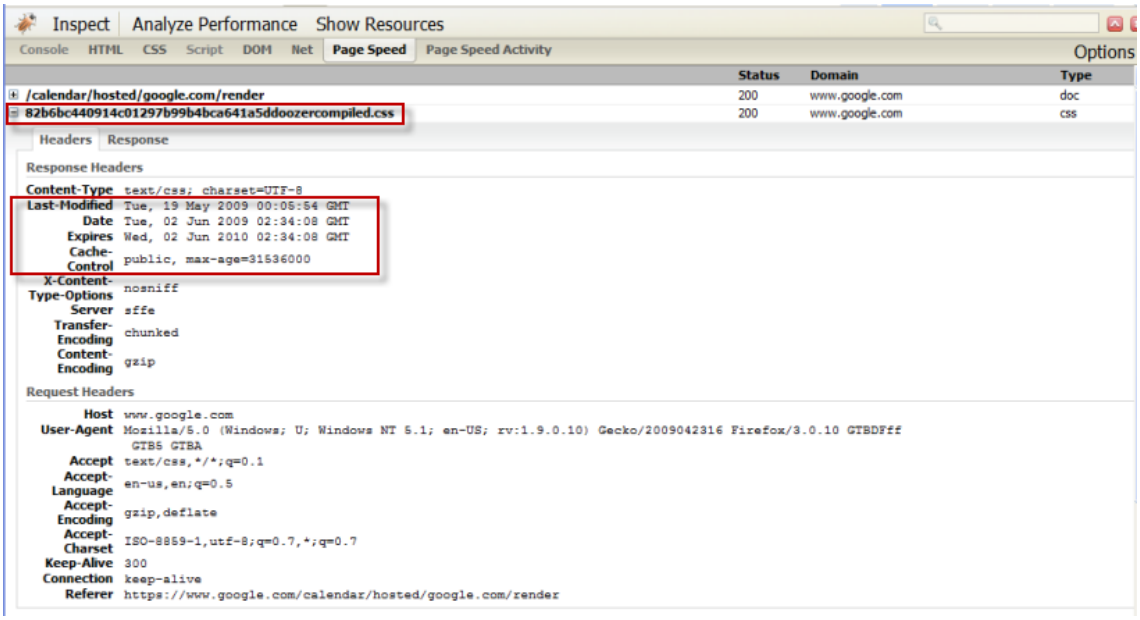


Abbildung 8: Cache-Header

Es wird also ein Last-Modified-Header gesetzt, der in der Vergangenheit liegt, das Datum wird geliefert und ein Expires-Header besagt, dass die Datei erst in einem Jahr wiedergeladen werden muss. Muss Google die Datei allerdings ändern, reicht ein geänderter Buchstabe in der Datei, um alle Browser davon zu überzeugen, dass es sich um eine völlig neue Ressource handelt, die er noch nie gesehen hat.

FORMEN

Es war klar, dass die Kommunikation auf JSON [29] basieren sollte. So war der Schritt zu einer definierten Form in der Datenbank nicht mehr groß. Anstatt für jeden Wettbewerbstyp eine eigene Reihenfolge im Code zu fixieren, wurde eine Verwaltung geschaffen, die die Reihenfolge und den Typ dieser Felder definieren kann. Die Flexibilität ist gegeben durch die freie Auswahl, welche Felder aus der Tabelle benutzt werden sollen sowie durch gewisse "Spezialfelder", die bei der Auswertung dem JavaScript über JSON -Markierungen mitgegeben werden. Dadurch ist es dem Client möglich, zwischen den Typen zu unterscheiden und für Felder wie z.B. dem Rank eine gesonderte Behandlung zu aktivieren.

DYNAMIC TEMPLATE GENERATION: TEMPLATES

Was versteht man unter Dynamic Template Generation? Ich habe diesen Begriff gewählt, da er meiner Meinung nach am besten ausdrückt, dass man mit einem WYSIWYG - Editor diese Templates erstellen und bearbeiten kann. Man kann Fenster erstellen und in Größe und Position verändern. Damit wird den Administratoren des Systems ermöglicht, nach eigenem Gutdünken das Layout zu verändern. Die Benutzer können dann dieses Template nehmen und nach ihren Bedürfnissen verändern. Templates sind Schablonen, die als Startpunkt für weitere Änderungen hergenommen werden können. Es gibt Patterns für Gebäude - warum also nicht auch für Computer-Interaktion?

Erste Versuche dafür gibt es z.B. von Jenifer Tidwell, 1998, mit ihrem Artikel *"Common Ground: A Pattern Language for Human-Computer Interface Design"* [30]. Sie beschreibt einige Möglichkeiten, die Oberfläche für verschiedene Datenanzeigen zu modifizieren:
Einerseits der *"Stack of Working Surfaces"* [31], wo in einer durch Tabs aufgeteilten Ansicht jede Ansicht einen eigenen Platz bekommt, der die mit den anderen Anzeigen interferiert. Man sieht von jeder Ansicht ausgehend sämtliche verfügbaren Möglichkeiten mit einem sichtbaren Label und kann mit minimalem Aufwand - am besten nur mit einem Klick - auf diese Umschalten. Allerdings kann man nicht mehrere Ergebnislisten auf einmal anzeigen.
Diese Möglichkeit wurde für die Livescoring-Ansicht verwendet, wo der einzige Sinn die Information über einen einzelnen Teil des Rennens ist.

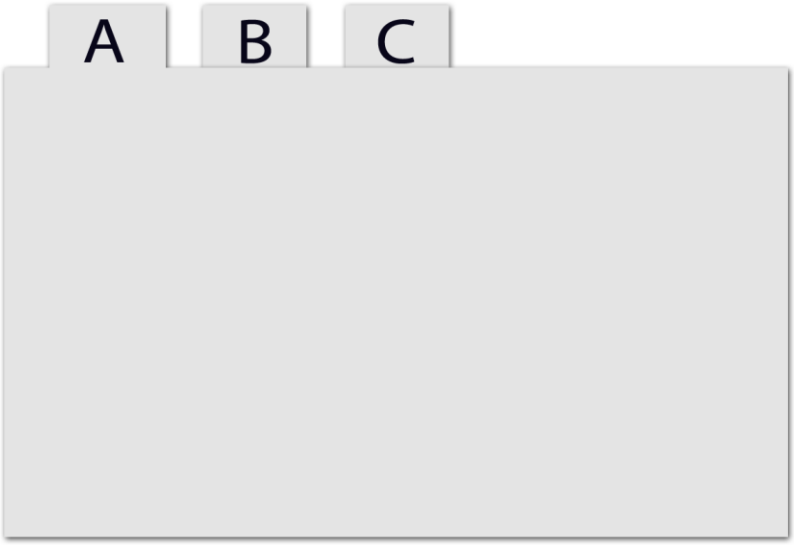


Abbildung 9: Stack of Working Surfaces (Tidwell)

Sie geht weiterhin auf *"Tiled Working Surfaces"* ein, wo mehrere visuelle Informationen sich eine Oberfläche teilen und genug Platz für jede Information ist. Als Beispiel werden Zeitungen, HTML frames, Toolbars genannt. Das Problem ist: Wie sollen diese Arbeitsoberflächen organisiert werden?

Diese Methode gibt Benutzern einen leichten Zugang zu mehreren Arbeitsoberflächen – sie macht immer alle Oberflächen sichtbar, und kann die Größenverhältnisse beeinflussbar machen, um den Informationen den richtigen Rahmen zu bieten oder genug Platz zu schaffen.

Abbildung 10: Tiled working surfaces

Als dritte Möglichkeit zeigt sie *"Piles of Working Surfaces"* [32]. Dabei handelt es sich um ein Fenstersystem, wie man es von Mac oder Windows-Systemen kennt - von einem physischen Tisch oder Pinnboard. Hiermit können wieder viele Arbeitsoberflächen leicht zugänglich gemacht werden, auch wenn nicht genug Platz ist, um alle vollständig zu zeigen.
Jede Oberfläche kann komplett unabhängig von den anderen positioniert und skaliert werden und das Bedürfnis, die Position und Skalierung der Fenster selbst zu bestimmen, kann leicht erfüllt werden. Sie empfiehlt, nur einen Layer zu bedienen, auf denen die Windows um die Anzeige konkurrieren können, und keinen weiteren Layer wie z.B. Alert-Fenster oder modale Fenster zu inkludieren. Ich möchte hinzufügen, dass diese Empfehlung aus dem Jahr 1999 stammt und dass die erwähnte Problematik bei guter Kennzeichnung bzw. Differenzierung der Layer heute kein Problem mehr darstellen sollte.

Um die Bedienung einfach zu halten, wurden bereits bekannte Muster verwendet. In weiterer Anlehnung an "About Face" von Alan Cooper [33] wurde das bereits bekannte "Fenster" - Muster verwendet. So können diese Fenster einen Typ erhalten und über diesen Typ bestimmte Informationen anzeigen, verschoben werden und ihre Größe kann angepasst werden - wie ein "Pile of Working Surfaces" von Tidwell.

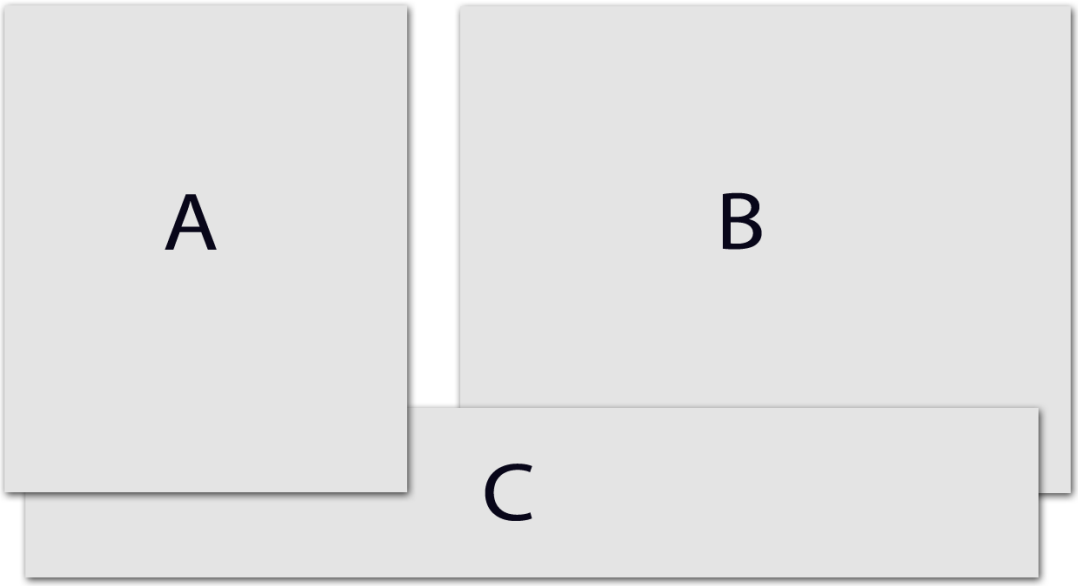


Abbildung 11: Pile of working Surfaces

Eine Zusammenfassung von Benutzererwartungen liefern Perzel & Kane, 1999, in ihrem Empfehlungsschreiben "*Usability Patterns for Applications on the World Wide Web"* [34]. Mit dem Pattern *"What They See is All They Get"* beschreiben sie das Problem, wie man dem Benutzer alle Informationen zur Verfügung zu stellt, die er sehen muss.

Als Probleme nennen sie:

- Benutzer scrollen im Internet lange Seiten oft nicht
- Jeder Benutzer hat eine andere Bildschirmaufteilung
- Benutzer erwarten ein Windows-Desktop-Interface mit Fehlerbehebungsfunktionen
- Fehlerseiten oder Fehlermeldungen können die Benutzer von der Anwendung vertreiben
- Die Textgröße ist abhängig von den Einstellungen des Endbenutzers und von der Verfügbarkeit der Schriftarten

Auch wenn diese Informationen schon über 10 Jahre alt sind und die erwähnten Oberflächengrößen in Ordnung von 640x480 angegeben werden, sind einzelne Punkte interessant zu diskutieren. So sind wir über die Benutzungsstatistiken ebenso schon auf den zweiten Punkt gekommen. Der erste Punkt besagt, dass mindestens die ersten Plätze von Haus aus sichtbar sein sollten. Die Erwartung einer Windows-Desktop-Oberfläche (schon 1999!) zeigt, dass die Oberfläche mit ähnlichen Interaktions-Mustern ausgestattet sein sollte wie übliche Desktop-Anwendungen (Resize, Move).

In einer neueren Abhandlung über *Interaction Patterns* von Welie und Troetteberg von 2000 wird ein *Grid Layout* vorgestellt [35], das Welie in einer späteren Version auf seiner Website verfeinert hat [36]. Das Bedürfnis vieler Benutzer, mehrere Objekte in einer klar organisierten Struktur zu sehen, wird hervorgehoben. Der Benutzer will die Zeit minimieren, in der man Objekte auf dem Schirm scannen bzw. lesen kann. Die Präsentation kann also kompakt sein, aber dennoch klar, freundlich und lesbar.

Die Anzahl der verwendeten Reihen und Spalten sollte demnach minimiert werden, und Objekte, die auch nicht unbedingt miteinander korrelieren, sollten an ähnlichen Ausrichtungslinien sitzen. Daher wird beim Erstellen und Einrichten der Templates sowie beim benutzerdefinierten Verändern ein Grid mit der Größe von 10px verwendet, um das Einhalten dieser unsichtbaren Ausrichtungslinien zu vereinfachen.

DYNAMISCHE DATENANZEIGE

Was sind eigentlich dynamische Daten? Dynamisch steht für Veränderung, Daten für eine allgemeine Menge an Informationen. Die meisten Projekte dieser Art haben etwas gemeinsam, und je generischer das Projekt, desto mehr lernt man für andere Zwecke.

Dieses Projekt sieht teilweise sehr einfach aus, teilweise sehr komplex. Es hat aber vor allem etwas mit vielen anderen Projekten zu tun die Daten dynamisch nachladen - die gleichen Mechanismen gelten für viele Websiten, die Informationen anzeigen. Die Lösung ist so generisch, dass damit viele Feeds gefüttert werden könnten - und mit wenig Umbauarbeit z.B. auch ein Chat oder Prüfungsergebnisse angezeigt werden könnten. Auch Livefeeds wie der Facebook-Newsfeed o.ä. benutzen ähnliche Systeme. Dynamische Daten sind das Prinzip, auf denen alle CRUDs und RIAs aufbauen. Man sieht also: Erfahrung in diesem Gebiet eröffnet einem viele Möglichkeiten, einen Beruf zu ergreifen.

Seit der Entwicklung des öffentlich zugänglichen Internets in Kombination mit HTML-Seiten 1995 wurde das Internet mehr oder weniger wie eine Sammlung von Buchseiten benutzt - mit Links, um von einer Seite zur nächsten zu springen [37]. Die Vernetzung der Computer war ein technologisches Wunder. Über die Zeit stiegen die Ansprüche der Internet-Benutzer, was zu weiteren Technologien und Weiterentwicklung von HTML führte. 1999 gab das W3C ein Empfehlungsschreiben heraus, um die übliche Kaffeepause während dem Laden einer Webseite zu vermeiden. Ein Teil dieser Empfehlung war die Verwendung von Stylesheets (CSS, XSL), ein anderer der (fehlgeschlagene) Versuch, mit HTTP/1.1 Headern die maßlose Benutzung von IPv4-Adressen zu verhindern [38].

Da das W3c als internationales Industrie-Konsortium bestehend aus dem MIT Laboratorium für Computerwissenschaften aus den USA [39], dem nationalen Forschungsinstitut für Computerwissenschaft und -kontrolle aus Frankreich [40] und der Keio-Universität aus Japan [41] bestand, hatte diese Empfehlung (fast vollständig) Erfolg - viele folgten den Vorschlägen. Die Verwendung von CSS bedeutete, dass Bilder jetzt mit Styleerklärungen ersetzt werden konnten - die Verwendung von PNG, dass die Bilder (teilweise) kleiner wurden. Die Verwendung von GIFs zur Anzeige von Bildern war damals vorherrschend, auch BMP wurde noch vereinzelt eingesetzt. Ein Stylesheet kann auch zur Beschreibung von mehreren Seiten verwendet werden - noch

ein Fortschritt, der eine Verbindung zum Webserver ersparen konnte.
Das Web war immer schon "Stateless" konzipiert - das bedeutet, der Client und der Server kommunizieren über einmalige Anfragen, die keine Verknüpfung mit der vorherigen haben müssen. Die ersten Anwendungen schufen neue Anforderungen. Beispielsweise gingen 1995 Amazon.com und Ebay online, 1996 war die erste Suche mit Hotmail möglich, 1997 Übersetzungen mit Babel Fish und 1998 die momentan populärste Seite, die für viele den Inbegriff des Internets darstellt: Google.

Die ersten Online-Versandhäuser schufen ein Bedürfnis, das es zuvor nicht gab: Der Client musste wiedererkannt werden. Also wurden Cookies eingeführt - erst 1997 vom W3c als "HTTP State Management Mechanism" standardisiert [42]. Damit wurde ein Hybrid-Status eingeführt, der sich bis heute generell[2] nicht geändert hat - die HTTP-Anfragen sind immer noch ohne Zustand, aber der Server kann den Client über Informationen identifizieren, die dieser ihm bereitstellt.
Dafür gibt es verschiedene Methoden. Zum Beispiel kann wie erwähnt der Client als Cookies eine ID und eine Authentifizierung mitschicken - ebenso kann er diese Daten als POST-Daten mitschicken oder im Querystring als GET übergeben. Der Mechanismus bleibt derselbe. Es benötigt zusätzliche Methoden auf der Serverseite, um den Benutzer zu authentifizieren - denn es könnte ein zweiter Client komplett zufällig dieselben Informationen schicken. Üblicherweise wird also serverseitig die IP des Clients in diese Information abgespeichert, oder gleich die Information mit Hilfe der IP verschlüsselt - was wiederum Probleme bei wechselnden IPs gibt. In dieser Arbeit wird eine Anwendung vorgestellt, die keine Authentifizierung benötigt, da sie für alle Endbenutzer die gleichen Informationen zur Verfügung stellt, daher sind diese Überlegungen nicht weiter von Belang.

[2] Ausnahmen bestätigen die Regel: WebSockets [141] funktionieren anders. Comet [142] ist nur eine alternative Verwendung von AJAX-Technologie.

Wie beeinflusst also Ajax die Kommunikation?

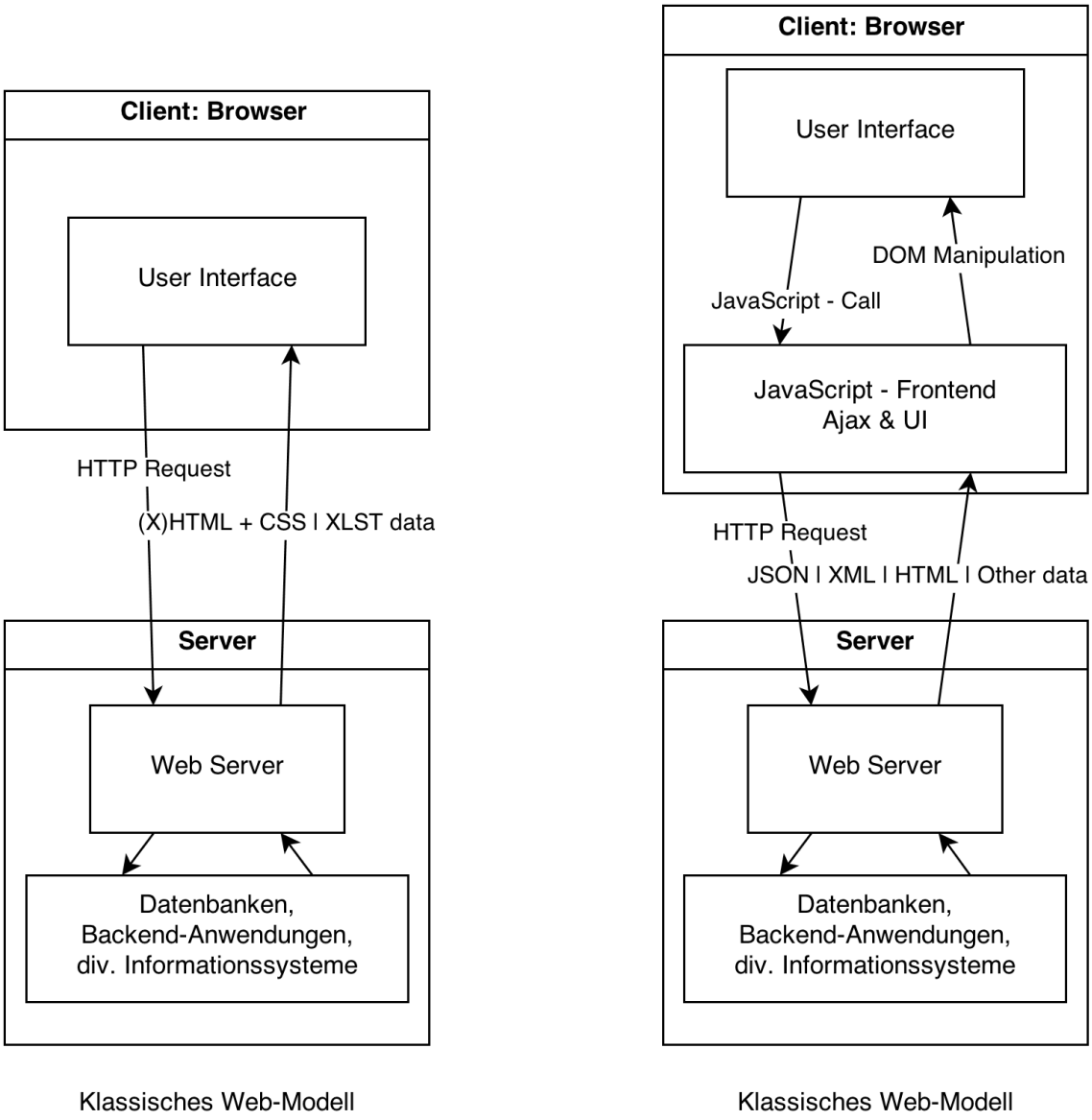


Abbildung 12: Kommunikation über Ajax

Es gibt also einen zusätzlichen Layer auf der Clientseite: Die JavaScript-Verarbeitung.

JavaScript hat sich seit Beginn seiner Existenz vom hässlichen Entlein zum glitzernden Schwan entwickelt. Begonnen hat JavaScript als ECMAScript im Netscape Navigator 2.0 - einem der ersten populären Browser überhaupt. 1997 wurde ECMAScript vom W3c standardisiert - erst zwei Jahre, nachdem der Internet Explorer 3 als erster JavaScript - fähiger Browser von Microsoft auf den Markt kam. Als Skriptsprache ist es

interpretiert ausgeführt, das bedeutet, dass der Code per Standard Zeile für Zeile ausgewertet wird und nicht vorkompiliert.
Doch selbst das ist nur teilweise wahr - so kompiliert die V8-JavaScript-Engine von Google den Code vor [43]. Man sieht also, JavaScript ist schon sehr leistungsfähig - was immer aufwändigere Web-Anwendungen ermöglicht.
Erst 2010 veröffentlichte Google "Google Instant", wo Benutzern über Ajax Suchvorschläge geliefert werden [44]. Ein Tastendruck reicht, und die ersten Vorschläge, was man suchen könnte, werden angezeigt [45].

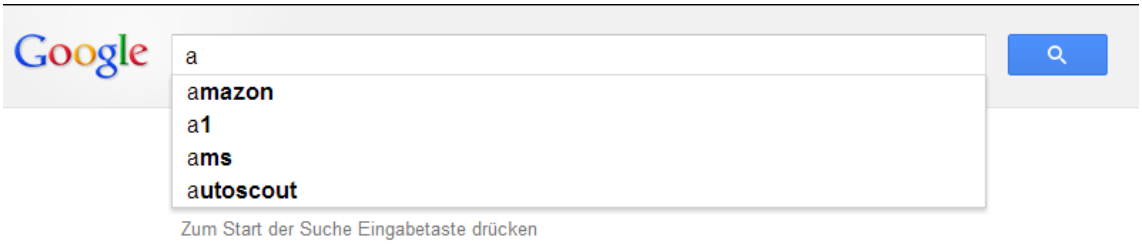


Abbildung 13: Google Instant

Google Instant war die erste berühmte Anwendung von dynamischen Daten. Über eine Verbindung werden von einem Server Daten abgefragt, die dem Benutzer anschließend angezeigt und für Interaktion aufbereitet werden.

Die Definition von Ajax bzw. DHTML wurde vom W3c erstmals 2006 herausgegeben [46]. Ein paar Jahre hat es gedauert, bis diese Empfehlungen auf Websiten und in Browsern implementiert wurden. In einer Präsentation von IBM 2006 wird noch die Frage gestellt: "What is AJAX and why do I care?" [47]. Die Gründe werden gleich geliefert - mittels Ajax kann man:

- Rich Internet Applications bauen
- die Interaktion im Internet dynamisch bauen
- die Perfomance verbessert werden
- Echtzeit-Updates geladen werden

ohne dass Plugins benötigt werden.

Als Probleme werden genannt:

- Der Benutzer weiß nicht, dass die Seite aktualisiert wird
- Der Benutzer bemerkt die Aktualisierung nicht
- Der Benutzer kann die neue Information nicht finden
- Der Fokus kann sich unerwartet ändern
- Der Zurück-Knopf funktioniert nicht mehr wie erwartet
- Spezifische Zustände können nicht mit einem Bookmark versehen werden

In den letzten 7 Jahren hat sich einiges verändert. So rechnen Benutzer inzwischen damit, dass die Seite auf Eingaben sofort reagiert - sie erkennen Updates und finden die neue Information. Moderne Frameworks sorgen dafür, dass der Fokus nicht verändert wird, es gibt Möglichkeiten, den Status aus Bookmarks zu lesen und den Zurück-Knopf wieder seiner Bedeutung zuzuführen.
Ein sehr imposantes Beispiel, das (unter anderem) von dynamische Daten lebt, ist Google Maps.

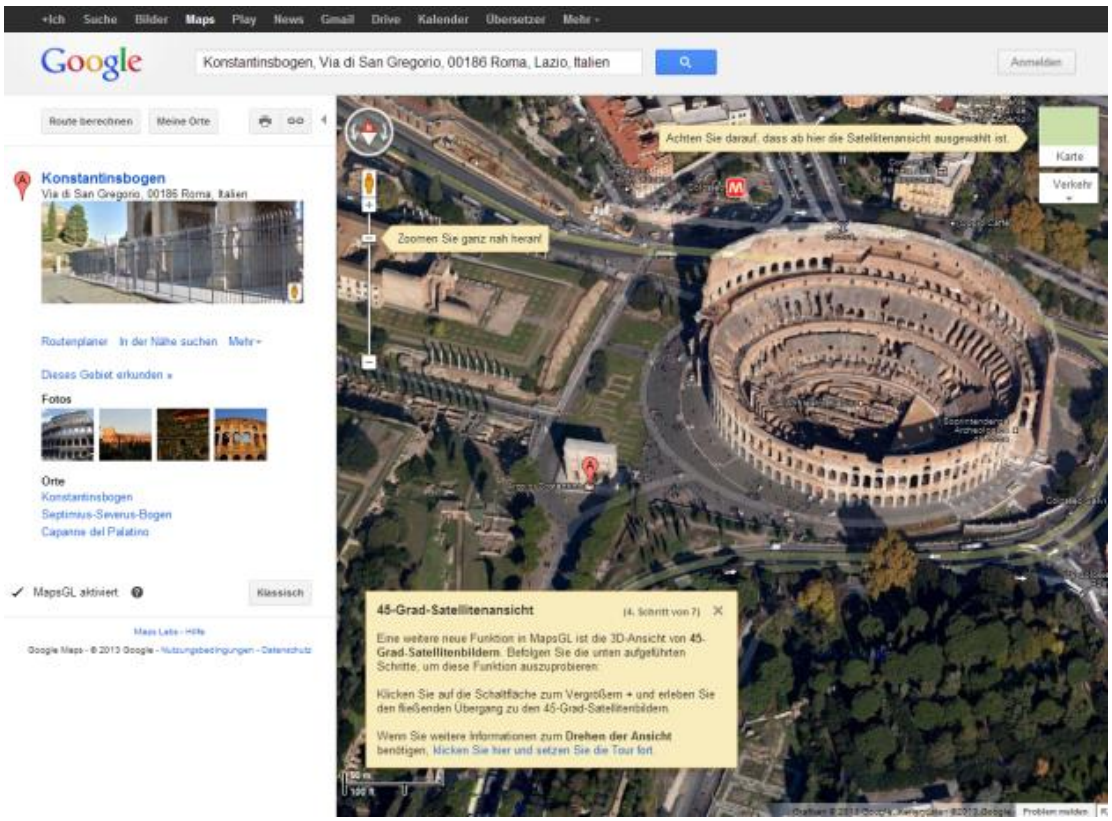


Abbildung 14: Dynamische Daten in Google Maps

ÜBERBLICK ÜBER VERFÜGBARE SPRACHEN UND METHODEN

CLIENTSEITIGE RENDERENGINE: XML - XSL - XSLT

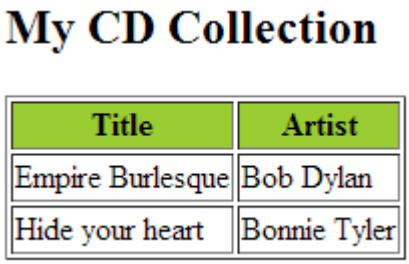
Laut wcschools.com unterstützen alle größeren Browser XML mit XSL-Transformation [48]. Firefox ab Version 3, Internet Explorer ab Version 6, Chrome seit Version 1, Opera seit Version 9 und Safari seit Version 3. Die Erstellung von Dokumenten mit XSLT - gefüllt mit XML - Daten und transformiert von XSL-Transformations [49] - ist auch denkbar einfach. Hier ein Beispiel von der w3schools-Seite:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
     <cd>
          <title>Empire Burlesque</title>
          <artist>Bob Dylan</artist>
          <country>USA</country>
          <company>Columbia</company>
          <price>10.90</price>
          <year>1985</year>
     </cd>
     <cd>
          <title>Hide your heart</title>
          <artist>Bonnie Tyler</artist>
          <country>UK</country>
          <company>CBS Records</company>
          <price>9.90</price>
          <year>1988</year>
     </cd>
</catalog>
```

Codebeispiel 1: XML - Code (Beispiel von wcschools.com [50])

Hier wird also in einer leicht zu lesenden Form ein Katalog von CDs erstellt. Leicht zu lesen, leicht zu editieren, sowohl für Menschen als auch für Maschinen, da das Format klar definiert ist. Der einziger Nachteil ist die relativ umfangreiche Beschreibungsform, die sich aber durch Abkürzungen der Bezeichnungen wieder im Traffic-Verbrauch reduzieren lässt.

Mit einer Transformation, einer XSLT-Anweisung, werden diese Daten in HTML verwandelt:



My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler

Abbildung 15: XML-XSLT Beispiel

Und so sieht die XSLT - Transformationsanweisung (Extensible Stylesheet Language Transformation) aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Codebeispiel 2: XSLT-Beispiel von w3schools.com

Woraus folgender HTML-Code wird:

```
<h2>My CD Collection</h2>
<table border="1">
<tbody>
  <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
  </tr>
  <tr>
    <td>Empire Burlesque</td>
    <td>Bob Dylan</td>
  </tr>
  <tr>
    <td>Hide your heart</td>
    <td>Bonnie Tyler</td>
  </tr>
  </tbody>
</table>
```

Codebeispiel 3: Resultierender HTML-Code v. XSLT-Beispiel

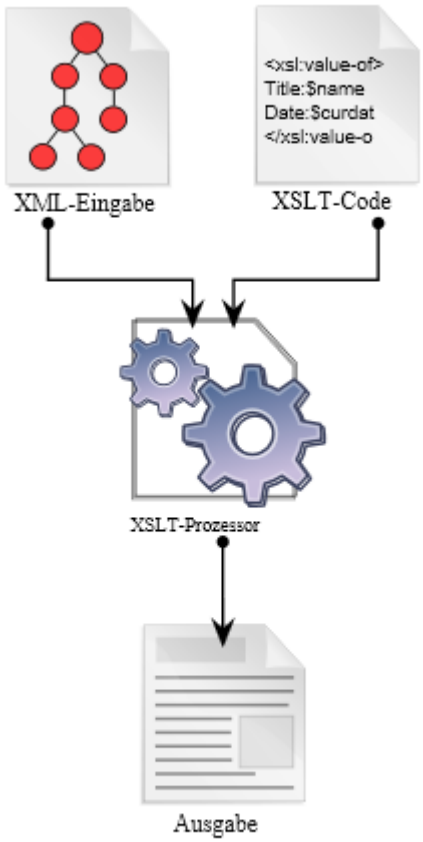


Abbildung 16: XSLT Erklärt [137]

In Zeile 2 ist ersichtlich, dass es sich bei diesem Beispiel um die XSLT-Version von 1999 handelt, die weitverbreitetste Version. XSLT kombiniert also XML-Eingaben und XSLT-Code in einem XSLT-Prozessor (den alle gängigen Browser Integriert haben) in HTML-Code. Für manche Anwendungsgebiete würde sich XSLT also einfach wie ein Templating-System für HTML verhalten - und immer noch CSS benötigen, um die Anzeige anzupassen. Die Hauptaufgabe für XSLT ist laut SelfHTML die Transformation von XML-Daten in HTML [51]. Wo ist also der "echte" Vorteil von XSLT? Wichtige Anwendungsgebiete sind z.B. POP (Presentation Oriented Publishing) und MOM (Message Oriented Middleware) [52].

So kann für POP die gezeigte Information komplett an die Ansprüche angepasst werden, z.B. aus den Daten können einzelne relevante Informationen gezogen werden, die dann basierend auf den Bedürfnissen der Benutzer gefiltert werden.
Hierfür gibt es Funktionen wie if, when, choose und for-each [53] [54]. Die Ergebnissprache muss auch nicht HTML sein, sondern kann durch ein stilbezogenes Markup ersetzt werden [55].
Für MOM, was auch dieses System wäre (Client-Browser-Anzeige), wäre XML eine Möglichkeit der Übertragung. Der Zwischenschritt über XSLT wäre hier allerdings ein zusätzlicher Aufwand, da aufgrund der Beschaffenheit der Daten sowieso noch ein JavaScript-Layer zwischen dem Laden und Verarbeiten der Daten liegen muss.

XSLT hat allgemein eine sehr schwache Verbreitung. Teile dieser Gründe liegen in den Anfängen, wo mit dem Internet Explorer 5 1998 ein XSLT-Prozessor mitgeliefert wurde, der nicht mit der Spezifikation von 2000 kompatibel war und damit für viele Probleme sorgte [56] [57]. Auch heutzutage gibt es wenige Beispiele für Websiten, die mit Clientseitigem XSLT geschrieben sind. Serverseitig gibt es Lösungen wie Apache Cocoon [58] oder das umfangreiche Content Management System Symphony [59].
Kritik gibt es für die zusätzliche Verarbeitungsstufe (Datenbank -> Verarbeitungssprache -> XML -> XSLT -> HTML) gegenüber direktem HTML (Datenbank -> Verarbeitungssprache -> HTML), aber auch für die formelle Art, wie Anweisungen in XSLT geschrieben werden müssen [60] [61] [62] [63] [64].

Zusammenfassend lässt sich sagen: XSLT hat seinen Aufgabenbereich in Situationen, wo Versatilität und Korrektheit die wichtigsten Punkte sind. Das W3C-Konsortium selbst empfiehlt die Verwendung von "CSS, wo man kann, XSL, wenn du musst", mit dem Hauptgrund dass CSS einfacher zu lernen und anzuwenden ist [65].

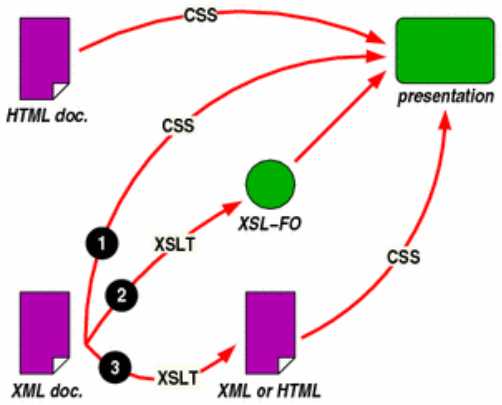


Abbildung 17: Zusammenhang CSS & XSL, w3c.org [65]

CLIENTSEITIGE RENDERENGINE: DOCUMENT OBJECT MODEL

Document Object Modeling ist das, was eine statische HTML-Seite zu einer dynamischen Seite macht [66]. Es handelt sich dabei um eine Plattform und ein sprachenneutrales Interface, mit dem Programme und Skripten auf den Inhalt von HTML-Seiten zugreifen und ihn auch verändern können. Das Dokument kann auch mittels Informationen, die im Dokument gefunden werden, weiter angepasst werden. Gängig ist der Begriff "Dynamic HTML", der dazu benutzt wird die Kombination aus HTML, CSS und Skripten zu bezeichnen, mittels deren man Dokumente animieren kann. Seit der Entstehung des Begriffs "Web 2.0", ca. 2004, wurde das, was unter XMLHttpRequest seit ca. 2000 funktionierte, unter dem Namen "AJAX" neu bekannt. Ajax ist ein Akronym für "Asynchronous JavaScript and XML" und hat für ein breiteres Verständnis von asynchroner Kommunikation gesorgt. Um die Anzeige einer (X)HTML - Seite anzupassen benötigt man CSS. Welche Version von CSS wird denn mobil unterstützt?

Feature	Safari on iOS	Android Browser	Google Chrome	Amazon Silk	BlackBerry Browser			Nokia Browser		Internet Explorer		Opera Mobile	Opera mini	Firefox	webOS Browser	
Platform	iPhone, iPad	Phones & Tablet	Android 4.0+	Kindle Fire	Phones	BB10	Tablet	MeeGo - N9	Symbian	Windows Phone 7.5	Windows 8	Android & Symbian	Java,iOS Android	Android, MeeGo	HP Phones	HP TouchPad
CSS 3 Basic W3C Standard opacity, backgrounds, text effects, rounded corners	✓	✓	✓	✓	✓ 6.0	✓	✓	✓	✓ Anna+	✓	✓	✓	✓ partial	✓	✓	✓
CSS 3 Transforms 2D W3C Standard rotate, translate, scale, skew, matrix	✓	✓ 2.0+	✓	✓	✓ 6.0	✓	✓	✓	✓ Anna+	✓	✓	✓	✓ partial	✓	✓	✓
CSS 3 Transforms 3D W3C Standard scale3d, translate3d, Perspective, Backface	✓	✓ 3.0+	✓			✓	✓	✓			✓			✓ 11+		
CSS 3 Transitions W3C Standard Animations between two states	✓	✓ 2.0+	✓	✓	✓ 6.0	✓	✓	✓	✓ Anna+		✓	✓		✓	✓	✓
CSS 3 Animations W3C Standard Animations with keyframes	✓	✓ 2.0+	✓		✓ 6.0	✓	✓	✓	✓ Anna+		✓	✓ 12.1+		✓	✓	✓

Abbildung 18: CSS-Unterstützung in Mobilen Browsern [67]

Das bedeutet, dass sogar CSS3 in allen gängigen mobilen Browsern verfügbar ist - mehr, als für dieses Projekt benötigt wird. Die Möglichkeiten von CSS2 sind vollständig ausreichend für dieses Projekt [3].

[3] Für einen Vergleich von CSS zu CSS2, siehe [138], für den Vergleich von CSS2 zu CSS3, siehe [139].

Das bedeutet, die HTML - Seite kann mit einem Doctype unserer Wahl gestaltet und mit CSS2 die Anzeige angepasst werden.

Wie sollen nun die Daten geladen werden? Hier gibt es mehrere Varianten:

- Plain HTML
  Die Tabelle könnte bei jedem neuen Teilnehmer komplett neu geladen werden.
  Fällt aufgrund der Anforderung "geringer Traffic-Verbrauch" weg.
- XML
  Der (neue?) Inhalt der Tabelle könnte in XML übertragen werden.
- JSON
  Der (neue?) Inhalt der Tabelle könnte in JSON übertragen werden.

XML oder JSON heißt die Entscheidung also. Die persönliche Präferenz des Autors wäre JSON, aber - warum?

XML und JSON sind zwei sehr ähnliche Standards - mit vielen Leuten, die verschiedenen Meinungen vertreten.

JSON selbst wird von den Erstellern als "Fat-Free Alternative to XML" bezeichnet [68]. Die Offizielle Seite beschreibt JSON als Datenaustauschformat, beschrieben in ECMAScript 262 [19].

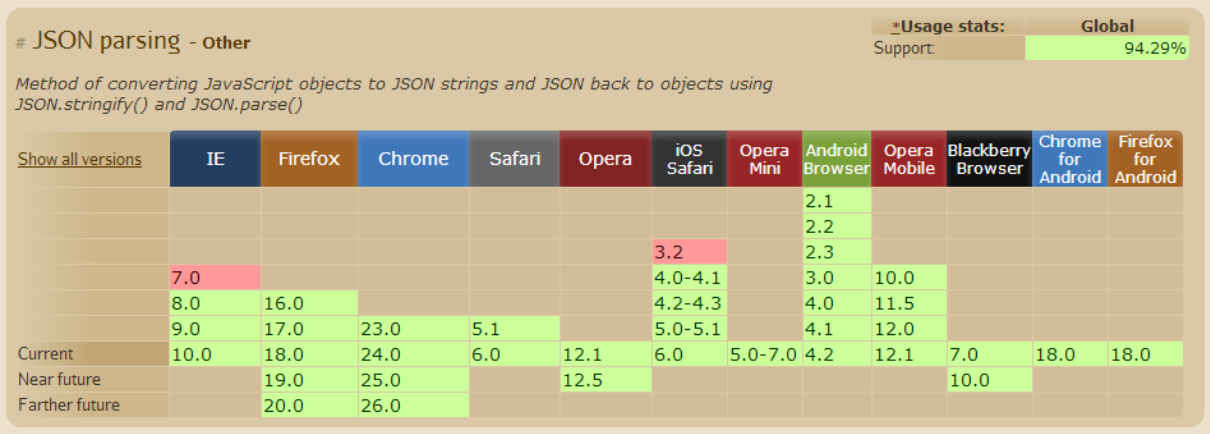
Eine Vergleichstabelle bringt eventuell mehr Licht in die Sache:

	XML	JSON
Lesbar von Menschen und Computern	**Ja**	Ja
Selbstdokumentierendes Format	**Ja**	Ja
Strenge Syntax	**Ja**	Ja
Niedriger Parsing-Aufwand	**Ja**	Ja (Simpler)
Encoding: Nativ Unterstützt von PHP	**Über Module** [69] [70]	Ja [71]
Parsing: Native Browserunterstützung	**Ja** [72]	Ja [73]

Diverse Quellen bezeichnen JSON als kleiner und schneller als XML [74] [75] [76], andere sagen die beiden sind sowieso gleich [77]. Der Vorteil "schneller" wird für JSON häufig erwähnt [78] [79] [80].
Peter-Paul Koch, Autor von "ppk on JavaScript" [81] und Ersteller von vielen Kompatibilitätstabellen auf quirksmode.org [82], hat sich intensiv mit diesem Thema beschäftigt, und kommt 2005 zu dem Schluss, dass JSON vielleicht noch nicht überall unterstützt wird und noch nicht so bekannt ist - daher bleibt er bei XML [83].

Wie sieht die Welt 2013 aus?

XML-Parser sind in allen Browsern nativ eingebaut. JSON-Parser auch - weil eval() jeden JavaScript-Code ausführt, und JSON JavaScript-Code ist. Wenn wir einen Schritt weitergehen und verlangen, dass JSON "echt" geparst wird, d.h. dass kein Code ausgeführt werden kann (Sicherheit!), ergibt sich folgendes Bild:



JSON parsing - Other

*Usage stats: Global

Support: 94.29%

Method of converting JavaScript objects to JSON strings and JSON back to objects using JSON.stringify() and JSON.parse()

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Blackberry Browser	Chrome for Android	Firefox for Android
								2.1				
								2.2				
						3.2		2.3				
	7.0					4.0-4.1		3.0	10.0			
	8.0	16.0				4.2-4.3		4.0	11.5			
	9.0	17.0	23.0	5.1		5.0-5.1		4.1	12.0			
Current	10.0	18.0	24.0	6.0	12.1	6.0	5.0-7.0	4.2	12.1	7.0	18.0	18.0
Near future		19.0	25.0		12.5					10.0		
Farther future		20.0	26.0									

Abbildung 19: JSON-Parsing Compatibility Table [84]

Alle gängigen Browser, auch mobil, unterstützen bereits Nativ das JSON-Parsen. Die zweite Möglichkeit wäre das Verwenden von jQuery oder einem anderen JSON-Parser. Der Unterschied besteht also in der Generation, wo PHP für die XML-Encodierung ein Modul benötigt oder eine extra geschriebene Funktion. Da die Daten aus einer MySQL-Tabelle gelesen werden, ist das Konvertieren zu JSON aus dem fertigen Array eine einzeilige Anweisung, die in jeder PHP-Umgebung[4] funktioniert. Hier "gewinnt" JSON gegenüber XML einen kleinen Vorsprung, weshalb JSON zur Übertragung verwendet wird.

[4] Seit PHP 5.2.0, 2006.

SILVERLIGHT

Silverlight ist ein proprietäres Tool von Microsoft. Es ist ein Plugin wie Flash, das installiert werden muss. Microsoft stellt selbst eine Übersicht zur Verfügung, wo Silverlight funktioniert. Da hier schon ersichtlich ist, dass es z.B. in keinem Safari-Browser unter Windows funktioniert und es auch bis jetzt noch keine veröffentlichte Version von Silverlight für mobile Geräte gibt, ist es nicht weiter relevant und wird nur der Vollständigkeit halber vorgestellt.

Operating System	Internet Explorer 10	Internet Explorer 9	Internet Explorer 8	Internet Explorer 7	Internet Explorer 6	Firefox 3.6+	Safari 4+	Chrome 12+
Windows 8 Desktop	✓*	-	-	-	-	✓	-	✓
Windows Server 2012	✓*	-	-	-	-	✓	-	✓
Windows 7	-	✓	✓	-	-	✓	-	✓
Windows 7 SP1	-	✓*	✓	-	-	✓	-	✓
Windows Server 2008 SP2	-	-	-	✓	-	✓	-	✓
Windows Server 2008 R2 SP1	-	✓*	✓*	-	-	✓	-	✓
Windows Vista	-	✓	✓	✓	-	✓	-	✓
Windows Server 2003, Windows XP SP2, SP3	-	-	✓	✓	-	✓	-	✓
Macintosh OS 10.5.7+ (intel-based)	-	-	-	-	-	✓	✓	-

* Supports 64-bit mode

Abbildung 20: Silverlight Kompatibilitätsliste

Für Silverlight gibt es vorgefertigte Klassen, mit denen dynamische Daten direkt angezeigt werden können, wie z.B. DataGrid [85].

JAVASCRIPT: PROTOTYPE (UND ANDERE FRAMEWORKS)

Prototype ist ein Framework [86] für Javascript, so wie z.B. auch mootools, Ext JS, YUI und über 1000 andere [87]. Es wird oft in der Verbindung mit script.aculo.us verwendet. Es soll hier stellvertretend für viele andere Frameworks besprochen werden, da es eines der ersten umfangreichen Frameworks für JavaScript war. Hier gilt noch viel mehr als bei jeder anderen Programmiersprache: Diese Wahl sollte jeder für sich selbst treffen. Warum hat der Autor als persönliche Preferenz und auch für dieses Projekt also jQuery gewählt?

Prototype hat z.B. einen Nachteil, es ist nicht in einen Namespace eingesperrt und verändert Objekte, die auch von anderen Modulen verwendet werden können, die daraufhin fehlschlagen können [88]. Als Hauptvorteil von jQuery zählt dessen Beliebtheit [89] [90] und damit verbunden die breite Verfügbarkeit von Plugins für viele Zwecke. Bei beliebten Frameworks ist es verständlicherweise leichter, Hilfe zu finden oder weitere Mitarbeiter.

Dennoch soll hier vorgestellt werden, wie in Prototype die Kommunikation mit dem Server aussehen würde. Die JavaScript-Datei:

```
var updater = Class.create({
    initialize: function(divToUpdate, interval, file) {
        this.divToUpdate = divToUpdate;
        this.interval = interval;
        this.file = file;
        new PeriodicalExecuter(this.getUpdate.bind(this),
this.interval);
    },

    getUpdate: function() {
        var div = this.divToUpdate;
        var interval = this.interval;
        var file = this.file;
        var oOptions = {
            method: "POST",
            asynchronous: true,
            parameters: "intervalPeriod="+interval,
            onComplete: function (oXHR, Json) {
                $(div).innerHTML = oXHR.responseText;
            }
        };
        var oRequest = new Ajax.Updater(div, file, oOptions);
    }
});
```

Codebeispiel 4: Prototype - Ajax Tutorial Code [91]

Hier wird eine Klasse[5] erstellt, die einen Konstruktor besitzt (`initialize`). Im Konstruktor wird ein periodischer Aufruf gestartet (`PeriodicalExecuter`), dem als Argumente der Zielcontainer, das Intervall und die abzufragende URL übergeben werden.

Die Funktion `getUpdate` ist eine Methode der Klasse. Sie greift auf die Funktion der Prototype-Klasse `Ajax` zu, der sie den Zielcontainer, die URL und ein Objekt mit Optionen übergibt. Der Code ist leicht lesbar und verständlich. Die Klasse muss jetzt noch generiert werden:

```
    document.observe('dom:loaded', function() {
        /*
        first arg   : div to update
        second arg  : interval in seconds
        third arg   : file to get data from
        */
        var visitorCounter = new updater('counter', 1,
'countVisitors.php');

        // Make first call so we get an immediate update after the
page is loaded
        visitorCounter.getUpdate();

    });
```

Codebeispiel 5: Prototype - Ajax Tutorial Code [91]

Beim Event `dom:loaded` wird also diese anonyme Funktion aufgerufen, die die Klasse erstellt und die Funktion gleich ein erstes Mal aufruft, um die Wartezeit zum ersten periodischen Aufruf zu verkürzen.

[5] Auch wenn es in JavaScript im engen Sinn keine Klassen gibt - zur leichteren Verständnis wird diese Bezeichnung benutzt.

Zum Vergleich, eine periodische Aktualisierung ohne ein Framework:

```
var request = false;

// Request senden
function setRequest() {
	// Request erzeugen
	if (window.XMLHttpRequest) {
		request = new XMLHttpRequest(); // Mozilla, Safari, Opera
	} else if (window.ActiveXObject) {
		try {
			request = new ActiveXObject('Msxml2.XMLHTTP'); // IE
5
		} catch (e) {
			try {
				request = new
ActiveXObject('Microsoft.XMLHTTP'); // IE 6
			} catch (e) {}
		}
	}

	// überprüfen, ob Request erzeugt wurde
	if (!request) {
		alert("Kann keine XMLHTTP-Instanz erzeugen");
		return false;
	} else {
		var url = "ajax_001.php";
		// Request öffnen
		request.open('post', url, true);
		// Request senden
		request.send(null);
		// Request auswerten
		request.onreadystatechange = interpretRequest;
	}

}
```

Codebeispiel 6: XMLHttpRequest erstellen ohne JavaScript - Framework [92]

Man sieht, der Code ist schon deutlich länger - und das nicht nur für die Erstellung des XMLHttpRequest - Objekts, sondern auch für das Abschicken dessen. Beim Abschicken wird das Attribut `onreadystatechange` auf die Funktion `interpretRequest` gesetzt, die noch nicht deklariert wurde.

```
// Request auswerten
function interpretRequest() {
	switch (request.readyState) {
		// wenn der readyState 4 und der request.status 200 ist,
dann ist alles korrekt gelaufen
		case 4:
			if (request.status != 200) {
				alert("Der Request wurde abgeschlossen, ist
aber nicht OK\nFehler:"+request.status);
			} else {
				var content = request.responseText;
				// den Inhalt des Requests in das <div>
schreiben
				document.getElementById('content').innerHTML =
content;
			}
			break;
		default:
			break;
	}

}
```

Codebeispiel 7: XMLHttpRequest auswerten ohne JavaScript - Framework [92]

Es muss in der Funktion also noch abgefragt werden, ob der `readyState` 4 und der Status 200 ist - danach darf der Content geladen werden. In den Container mit der ID "`content`" wird dann der HTML-Code geschrieben.

Aufgerufen muss dieser Code natürlich auch noch werden, dafür genügt ein Aufruf von `setInterval(setRequest, 1000)`.

Ohne Framework ist zumindest also ziemlich viel Code zu schreiben - und besonders flexibel sieht es auch nicht aus.

JAVASCRIPT: JQUERY

jQuery ist eine weiteres JavaScript - Framework, 2005 gegründet von John Resig. 2006 wurde jQuery beim BarCamp NYC angekündigt. Kurz danach wurde es unter MIT Lizenz gestellt, was es für alle Zwecke frei einsetzbar und veränderbar macht [93]. Es ist Plugin-basiert, was bedeutet, dass fast alle Anweisungen und Verwendungszwecke aus solchen bestehen [94]. Es gibt eine unzählbare Anzahl an Plugins für jQuery, die bekannteste Sammlung ist jQuery UI [95], die einige bekannte Werkzeuge von Desktop-Benutzeroberflächen für HTML umsetzen - mit Hilfe von JavaScript, CSS und HTML. jQuery Mobile ist ein Unterprojekt, das jQuery für mobile Geräte optimiert und auch Touchgesten unterstützt [96]. Über 90% der Seiten, die ein JavaScript - Framework verwenden, verwenden jQuery [97].

Viele geschichtliche Informationen - doch wie sieht der Code aus?

```
function loadData()
{
  $('#content').load("/daten.php");
}

setInterval(loadData, 1000);
```

Codebeispiel 8: Beispiel: Daten vom Server - jQuery

Und fertig.

In diesen wenigen Zeilen geschieht viel. Man könnte das ganze Programm auch in eine Zeile schreiben, wenn man wollte. Das wäre bei Prototype auch kein Problem - bei jQuery ist es allerdings natürlich, die Anweisungen so kurz zu schreiben.

$ ist bei jQuery das magische Symbol. Wenn ein Objekt mit $() umrandet wird, bekommt dieses Objekt alle jQuery-Methoden vererbt. Die verwendete Technik dafür nennt sich bei JavaScript prototyping - woher auch der Name für das bekannte Framework Prototype herkommt.

Es gibt also offensichtlich eine Funktion load, die eine Datei aufruft und das Ergebnis als HTML-Text in den Container schreibt. Doch was tut die Raute in der Klammer? jQuery ist definiert von CSS-Selektoren.
Über alle gängigen CSS-Selektoren - bis inklusive CSS3 und einigen eigenen Erfindungen können Elemente angesprochen werden [98]. Wenn also statt der Raute ein

Punkt in der Klammer wäre, würde die Anweisung `$('.content')` alle Elemente mit der Klasse content selektieren.

Zwei Container, die diese Klasse hätten, würden *beide seperat voneinander* die load-Funktion ausführen. Wenn wir das nicht wollen, können wir die Funktion auch einfach anders schreiben:

```
$.get('daten.php', function(data){

  $('.content').html(data);

});
```

Codebeispiel 9: Alternativbeispiel: Daten vom Server - jQuery

Der Code ist noch nicht wirklich länger, aber schon sehr viel vielseitiger - wir könnten den Code zuerst bearbeiten, durch Filter oder Fallunterscheidungen schicken und allgemein - einfach bearbeiten wie wir wollen.

Wie würde man jetzt auf JSON-Daten reagieren?
Es wird nicht viel komplizierter.

Wir gehen wie im jQuery - Tutorial von der folgenden PHP-Datei aus [99]:

```
<?php

echo json_encode(array("name"=>"John", "time"=>"2pm"));

?>
```

Codebeispiel 10: PHP-Datei mit JSON Daten

Um jetzt den Namen und die Uhrzeit in den Container zu schreiben, müssen wir das Codebeispiel wie folgt umändern:

```
$.get('daten.php', function(data){

$('.content').html("Name: " + data.name + "<br>" + "Zeit: " +
data.time);

});
```

Codebeispiel 11: JSON-Verarbeitung in jQuery

Man sieht also, jQuery hat eine eingebaute Unterstützung für JSON, die den Zugriff auf das rückgelieferte Objekt sehr vereinfacht.

SERVERSEITIGE VERARBEITUNGSSPRACHEN

Die serverseitige Verarbeitungssprache ist bereits gewählt (PHP). Das liegt an der Erfahrung und persönlichen Präferenz des Autors. Dennoch sollen hier die Alternativen mit ihren Vor- und Nachteilen besprochen werden.

ASP.NET

ASP ist das Komplementärprodukt von PHP von Microsoft. ASP steht für Active Server Pages und ist von der Syntax her ähnlich zu Visual Basic. Der Nachfolger von ASP ist ASP.NET, mit dem unter Verwendung von allen CLR-Kompatiblen Sprachen (z.B. Visual Basic, C#, J#) Seiten erstellt werden können. Unter Verwendung vom Apache-Modul mod_aspdotnet und mod_mono kann ASP.net auch unter Linux verwendet werden.
Ab Installation stehen in ASP viele Funktionen wie z.B. Bildverarbeitung und XML-Verarbeitung Standardmässig zur Verfügung. Es ähnelt also PHP unter Verwendung von größeren Frameworks, wie z.B. Zend. Der meistgenannte Vorteil von ASP.net ist die starke Integrierung mit der Visual Studio IDE [100]. Die Verbreitung hat sich stark geändert, während es 2008 noch mehr Jobs für ASP.net - Entwickler gab [101], gibt es 2013 deutlich mehr Jobs für PHP-Entwickler [102].

Es ist mit ASP.net auch möglich auf MySQL - Daten zuzugreifen, und es gibt auch freie Möglichkeiten, ASP.net - Seiten zu hosten - man muss also nicht mehr zwangsweise für eine Serverlizenz zahlen. Allerdings sind diese nicht immer hundertprozentig kompatibel und nicht so weit verbreitet. Laut W³Techs sind ca. 20% der Websiten mit ASP.net geschrieben [103]. Beim Vergleichen der beiden Sprachen scheint die jeweilige Seite ein Unwissen gegenüber der jeweiligen Gegenseite zu besitzen. Ob ASP oder PHP "besser" sind, kann (unter anderem Deswegen) mit ruhigem Gewissen als Glaubenskrieg bezeichnet werden [104] [105] [106] [107][6].

[6] Es gibt sogar Converter für PHP-Code zu ASP-Code, z.B. [140]

PHP: ZEND FRAMEWORK

Das Zend Framework, mittlerweile in Version 2 verfügbar, ist eine MVC - Implementation, die fertige Funktionen für Forms, Validierung, Filterung und Authentifizierung bereitstellt und damit die Entwicklungszeit von Web-Projekten reduzieren will [108].

Es steht hier wieder stellvertretend für viele andere Frameworks wie z.B. Symfony [109], yii [110] oder cakephp [111]. Da einer der wichtigsten Faktoren Geschwindigkeit war, und alle Frameworks von Haus aus einen starken Einfluss (bis Faktor 10) auf die Ausführungszeit haben, ist die Anwendung ohne ein Framework geschrieben worden [112].

JAVA SERVER FACES

Java Server Faces sind ein Web Framework für Java. Es folgt dem MVC - Prinzip. Mit globalen 4% ist Java nicht besonders weit verbreitet, dennoch benutzen so populäre Seiten wie msn.com, ebay.com, paypal.com oder aol.com Java. Unter den Top 1000 der beliebtesten Seiten sind sogar 25% der Seiten mit Java geschrieben - 29% mit ASP.net, der Anteil von PHP sinkt auf 59,8% [113].

Für Java Server Faces gibt es einige Erweiterungen wie z.B. RichFaces, dojofaces, Openfaces, ICEfaces. Auch Testumgebungen wie JSFUnit sind vorhanden, und auch direkte Integrierungen von jQuery z.B. mit jQuery4jsf. Es ist also eine komplette Entwicklungsumgebung mit einer nicht so kleinen Community wie man aufgrund der globalen Verbreitung annehmen könnte. JSF wird zur Entwicklung von Standard-Webanwendungen empfohlen, mit Erweiterungen als Rich Client Anwendung [114] oder CRUD (Seam Framework [115]) [116] , aber nicht für Rich Internet Applications [117] [118].
Mit JSF hat man wenig manuelle Kontrolle über die AJAX-Requests, da es als komponentenbasierendes MVC - Framework diese Aufgaben selbst übernimmt. Für so ein Projekt wie dieses wäre es vollkommen fehl am Platz - die Stärke liegt eher in der schnellen Erstellung von Web Interfaces [119], je nach Erweiterung sogar mit integrierter GUI-Entwicklung direkt in der IDE [120].

CACHE

Wieso eigentlich ein Cache? Ist es nicht genug, mit intelligenten Mechanismen so wenig Information wie möglich abzufragen? Wie sieht die Zugriffsstatistik aus?

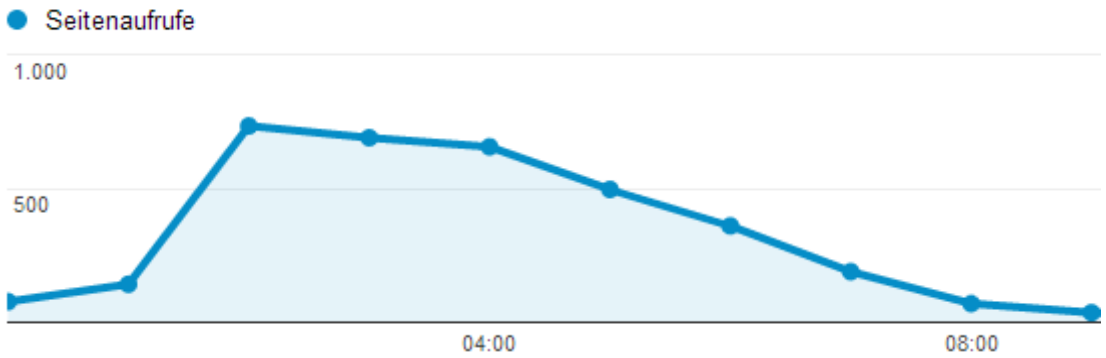


Abbildung 21: Zugriffsstatistik eines Live-Scoring Events

Wie in dem obigen Beispiel ersichtlich, steigen die Zugriffe bei so einem Event sprunghaft an und fallen dann wieder rapide ab. Bei einem mittelgroßen Event gibt es also 6-800 Seitenaufrufe - pro Stunde. Ca. 40% davon sind wiederkehrend (refreshes), das bedeutet, dass am Ende einer solchen Veranstaltung ca. 2000 Zuseher die Veranstaltung betrachten. Ein typischer billiger Webserver mit ~8 GB Ram und einem Quadcore verträgt ca. 100-200 gleichzeitige Seitenaufrufe, alles was darüber hinausgeht wird in eine Warteschleife gestellt. Sobald die Schleife voll ist, beginnt es problematisch zu werden und die Ladezeiten verlängern sich - was bei einem regelmäßigen Aktualisierungsintervall problematisch werden kann [7]. Wenn ein Server nicht gut konfiguriert ist und zu viele Zugriffe gleichzeitig erlaubt, steigt der Bedarf an RAM an, bis auf virtuellen (Festplatten-)Speicher zugegriffen werden muss. Wenn das passiert, kommt das System bei gleichbleibender Last mit dem Swappen nicht mehr nach und kann mit der Leistung einbrechen [121].

Die meisten Aufrufe wollen genau dasselbe sehen wie der vorherige Teilnehmer - daher bietet es sich an, ein Caching-System zu schreiben, das mit weniger oder keinen Datenbank-Zugriffen auskommt.

[7] Auch wenn das JavaScript sich clientseitig auf so einen Fall einstellt, wie später beschrieben.

FAZIT

Es gibt bereits viele Methoden, Frameworks und Hilfen, um dynamische Daten anzuzeigen. Frameworks sind aufgrund ihrer Beschaffenheit auf generische Probleme fokussiert oder haben oft gewisse Performance-Einbußen zu verzeichnen, wenn sie in ihren Möglichkeiten zu breit gefächert sind.

Was sind - nochmal zusammengefasst - die Anforderungen von dieser Anwendung?

- Geschwindigkeit (Caching)
- Trafficverbrauch
- Individualisierbarkeit
- Geräteunabhängigkeit

Da dieser Anwendungsfall sehr speziell ist und bisher in ähnlichen Variationen nur proprietär gelöst wurde, muss diese Anwendung von Grund auf neu gebaut werden.

Das einzige Framework, dass verwendet wird, ist das JavaScript-Framework jQuery, da hier der Performance-Nachteil deutlich von der gewonnenen Effizienz bei der Programmierung und bei der Fokussierung auf verschiedene Geräte übertönt wird. Für das CSS wird ein von jQuery-UI zur Verfügung gestelltes Template benutzt, serverseitig ein Wrapper für die Verbindung zur Datenbank, geschrieben von mir selbst.

PROJEKTZIELE

Das Ziel dieser Arbeit war eine Darstellung von dynamischen Daten, die keine übliche Form (Chat, Forum, etc.) wie schon bekannt darstellt, sondern eine neue Art und damit eine Herausforderung ist. Als Basis wurden mir echte Daten aus Skirennen zu Verfügung gestellt (Teilnehmerdaten sowie teilweise echte Ergebnisdaten).
In diesem Projekt werden die Ergebnisse zufallsgeneriert erstellt und dann in einer der gewählten Betrachtungsformen dargestellt.
So kristallisierten sich über den Projektverlauf mehrere Unterziele heraus, die im ersten Abschnitt kurz beschrieben werden sollen. Im zweiten Teil dieser Arbeit wird der theoretische Hintergrund sowie aktuelle Forschungen zu diesem Thema erläutert.

Da diese Arbeit als praktische Arbeit angelegt ist, ist das Hauptaugenmerk auf den dritten Teil - die praktische Umsetzung - gelegt. In dieser Arbeit sollen, wie in der Kurzfassung erwähnt, nur die Aspekte dieses Systems stehen, die einzigartig für dieses Projekt sind. So werden gängige Module dieses Systems wie die Verbindung zur Datenbank, das Bearbeiten von Formen etc. aufgrund der geringen Schöpfungshöhe nicht weiter behandelt.
Bei der Recherche zu dieser Arbeit konnte auf das Fachwissen von mehreren Leuten zurückgegriffen werden, die als Zeitnehmer für die FIS arbeiten. Dadurch entstanden einige Anforderungen, die dem System eine zusätzliche Komplexität verschafften und die Herausforderung erhöhten.

Als Inspiration diente das Livescoring der FIS selbst. Es zeigt die Daten direkt während dem Rennen an, ist jedoch sehr fehleranfällig. Es erfüllt nicht alle Anforderungen der Leute, die es brauchen (z.B. der Sprecher, der Trainer,..) und ist in Flash programmiert. Es wurde eine HTML - Version angekündigt, die bis Jänner 2013 nicht umgesetzt wurde.

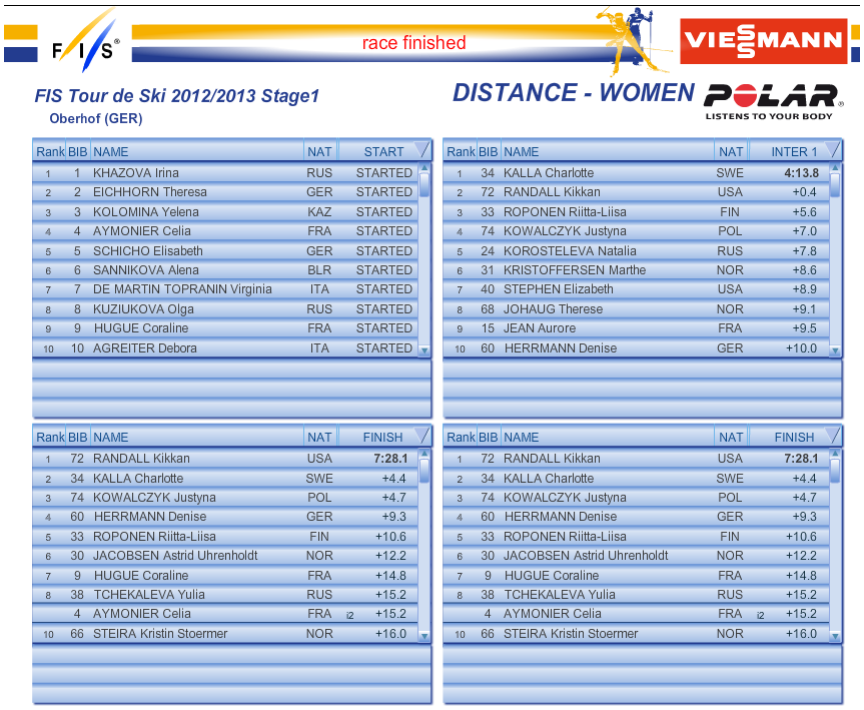
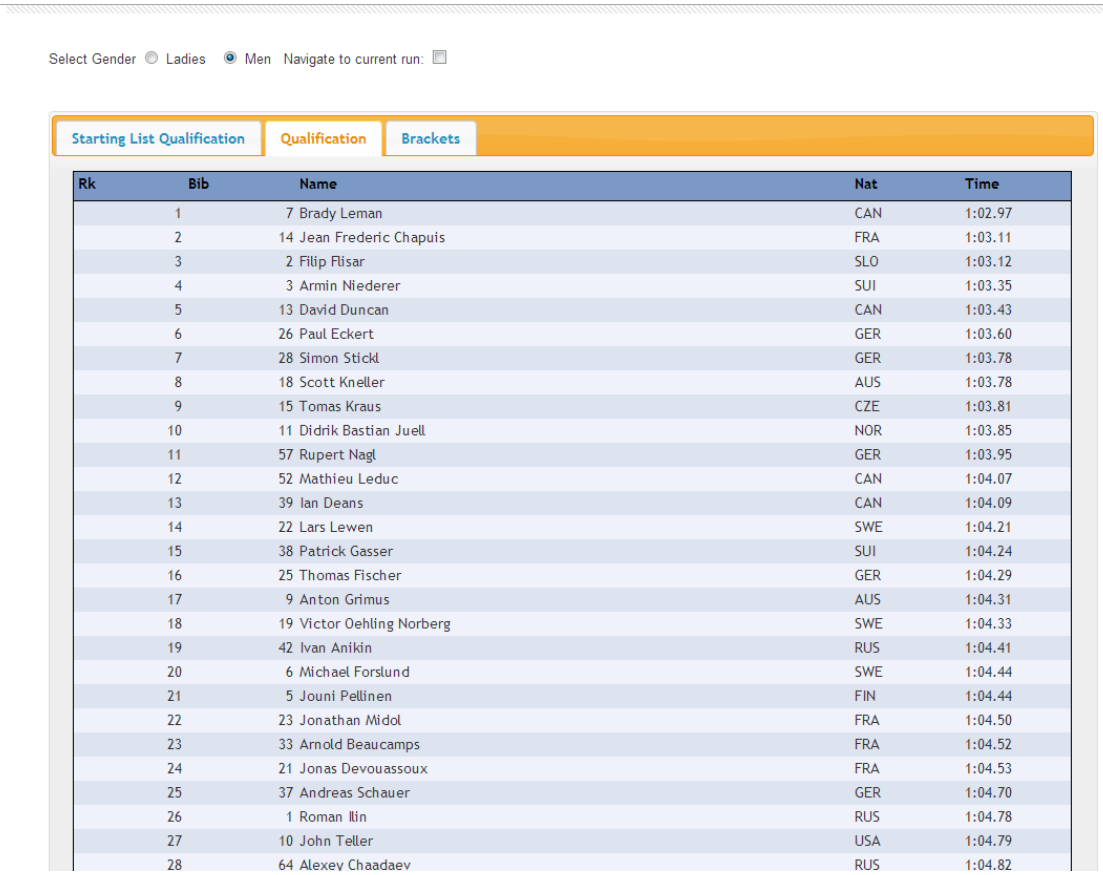


Abbildung 22: FIS-Livescoring

LIVESCORING

Was wird dieses Projekt simulieren? Es wird auf einer Website eine Übersicht angezeigt, die aktuellen Ergebnisse eines Skirennens sollen dort dargestellt werden. Dazu wird eine in Tabs unterteilte Anzeige verwendet - pro Teilbewerb dieses Rennens soll ein Tab verwendet werden. Es sollen sämtliche Ergebnisse hier nachsehbar werden. Ein Beispiel für einen gefüllten Tab des Livescorings:



Rk	Bib	Name	Nat	Time
1	7	Brady Leman	CAN	1:02.97
2	14	Jean Frederic Chapuis	FRA	1:03.11
3	2	Filip Flisar	SLO	1:03.12
4	3	Armin Niederer	SUI	1:03.35
5	13	David Duncan	CAN	1:03.43
6	26	Paul Eckert	GER	1:03.60
7	28	Simon Stickl	GER	1:03.78
8	18	Scott Kneller	AUS	1:03.78
9	15	Tomas Kraus	CZE	1:03.81
10	11	Didrik Bastian Juell	NOR	1:03.85
11	57	Rupert Nagl	GER	1:03.95
12	52	Mathieu Leduc	CAN	1:04.07
13	39	Ian Deans	CAN	1:04.09
14	22	Lars Lewen	SWE	1:04.21
15	38	Patrick Gasser	SUI	1:04.24
16	25	Thomas Fischer	GER	1:04.29
17	9	Anton Grimus	AUS	1:04.31
18	19	Victor Oehling Norberg	SWE	1:04.33
19	42	Ivan Anikin	RUS	1:04.41
20	6	Michael Forslund	SWE	1:04.44
21	5	Jouni Pellinen	FIN	1:04.44
22	23	Jonathan Midol	FRA	1:04.50
23	33	Arnold Beaucamps	FRA	1:04.52
24	21	Jonas Devouassoux	FRA	1:04.53
25	37	Andreas Schauer	GER	1:04.70
26	1	Roman Ilin	RUS	1:04.78
27	10	John Teller	USA	1:04.79
28	64	Alexey Chaadaev	RUS	1:04.82

Abbildung 23: Das Livescoring im Einsatz

Man kann hier verschiedene Aspekte bereits erkennen, die bei der Anzeige bedacht werden müssen:

- Die Ergebnisse müssen (im Normalfall) für Männer und Frauen getrennt angezeigt werden.
- Es muss ein System geben, dass erkennt in welchem Run sich die Teilnehmer momentan befinden, und die Anzeige darauf basierend wechselt

- In seiner simpelsten Form ist das Livescoring eine Tabelle, in der Textwerte stehen.

Wenn das System so simpel wäre wie auf diesem Beispiel ersichtlich, wäre das Thema einer Masterarbeit nicht würdig gewesen. Kommen wir also zur zweiten Anzeige, dem CIS.

CIS

Das CIS (Competition Information System) hat eine etwas diffizilere Aufgabe. Hier müssen mehrere Anzeigen auf einmal funktionieren, und diese sollten vorgegeben und dennoch modifizierbar sein.

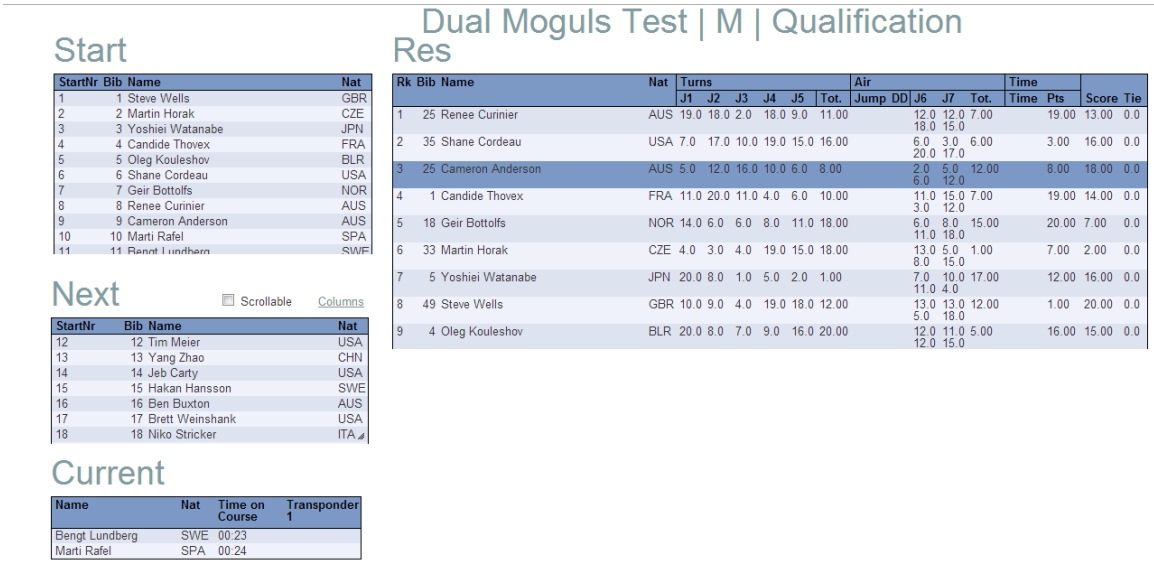


Abbildung 24: CIS

Hier sind wieder verschiedene Punkte zu bemerken:

- Es soll 1-x Fenster auf einer Seite geben
- Es gibt verschiedene Fenstertypen mit verschiedenen Anzeigeformen (Tabellenhead/Inhalt)
- Die Fenster können eine frei wählbare Überschrift haben
- Die Fenster können frei verschoben werden - vorgegeben und vom Client überschreibbar
- Die Fenstergröße kann frei eingestellt werden - vorgegeben und vom Client überschreibbar
- Der Abfragetypus der Fenster ist unterschiedlich:
    - Das Startfenster verändert sich selten, da es alle Starter für diesen Heat anzeigt. Wenn es sich verändert, müssen alle Daten ausgetauscht werden.

  - Das Next-Fenster ändert sich pro Run, da der jeweils nächste Startende angezeigt wird (Menge der Starter minus Current/Result). Da die Präsenz von Current und oder Resultfenster nicht garantiert ist, muss diese Information vom Backend kommen.
  - Das Current-Fenster zeigt alle Teilnehmer an, die gerade auf der Strecke sind. Diese Daten müssen immer aktuell sein und immer neu geladen werden. Das Time-On-Course Feld muss sich relativ von Client und Serverzeit eine möglichst genaue Zeit errechnen, die der Teilnehmer bereits auf der Strecke ist.
  - Das Ergebnisfenster hat eine fixierte Größe und ist auf "Scroll to last Competitor" gestellt. Das bedeutet, dass nach den ersten drei Plätzen so lange Teilnehmer ausgeblendet werden, bis der aktive Competitor sichtbar ist.
- Es gibt einen Link zur Spaltenverwaltung, bei der die aktuelle Spalteneinblendung überschrieben werden kann.

HOHE PERFORMANCE

Unter der Annahme, dass Daten von Skirennen angezeigt werden, wird das System einer eher unüblichen Belastung ausgesetzt: Über eine lange Zeit wird niemand auf das System zugreifen, dann steigen die Zugriffe innerhalb einer kurzen Zeit stark an.

Daher muss das System eine gewisse Peak-Abfederung haben. Als Ziel wurde hier eine flexible Caching-Lösung angestrebt, die mehrere hundert Benutzer auf einem einzigen Server bedienen können soll.
So sollen nicht nur die statischen Dateien nach den gängigen Web-Standards so selten wie möglich neu ausgeliefert werden, sondern auch die Daten von der Datenbank sollen wenn möglich nur einmal generiert werden. Das gibt natürlich ein Problem - wie bekommt man den Client dazu, zu wissen wann die Daten neu sind? Die Lösung dafür besteht in einem Versionierungssystem, das den Status des Systems überwacht.

Zusammengefasst ist das System naheliegender Weise "*Caching*" genannt worden.

ANZEIGEFORMEN: PRO WETTBEWERBSTYP

Im Zuge der Recherche für die Diplomarbeit kristallisierte sich heraus, dass die Ansprüche für jeden Wettbewerbstyp verschieden sind. So gibt es Wettbewerbsformen, die einen Run[8] pro Heat haben und welche mit 2 oder mehr. Es gibt Formen, wo alle Ergebnisse nebeneinander stehen sollen und Formen, wo manche Ergebnisse untereinander stehen sollen. Manche Felder müssen nur den Text anzeigen, der in der Datenbank steht, andere müssen Informationen aus einer anderen Tabelle holen oder eine Sonderaufgabe erfüllen, z.B. bei der Sortierung. Dafür wurde das System der "*Formen*" kreiert.

ANZEIGEFORMEN: PRO ZIELGRUPPE

Bei einem Event wie einem Skirennen gibt es verschiedene Zielgruppen: Die Sprecher der Fernsehstationen haben andere Bedürfnisse als die Teamtrainer oder die Punkterichter des Wettbewerbs. Daher müssen die gleichen Daten auf verschiedene Art dargestellt werden können. Um diesem Anspruch zu genügen und dabei eine hohe Flexibilität zu ermöglichen, wurden verschiedene Einheiten erstellt, die sich "Windows" nennen. So soll ein Template beliebig viele Windows haben, und in diesen verschiedene Informationen anzeigen können. Dafür wurde das System der "*Templates*" erstellt.

ADMINISTRIERUNG: TEMPLATES

Um die Templates mit ihren Funktionen administrierbar zu machen, muss ein eigenes Interface erstellt werden. Es sollte ermöglichen, Templates leicht zu erstellen, zu kopieren und zu verändern. Templates sollten pro Disziplin erstellt werden können und es sollte eigene Templates für die Qualifikationsrennen und für die Finalrunden geben. So soll man Templates erstellen können, die für alle Rennen der Disziplin "Moguls" verfügbar sind, z.B. ein Template "Trainer", "Sprecher", etc.

[8] Für die Begriffserklärungen: siehe Abkürzungsverzeichnis

ADMINISTRIERUNG: SPALTENAUFTEILUNG

Bei den *Formen* geht es nicht nur um die Spalten die angezeigt werden, sondern auch um deren Breite. Hier muss also ein System geschaffen werden, dass diese Breite leicht änderbar macht und dennoch für alle Anzeigebreiten gleichermaßen funktioniert.

ADMINISTRIERUNG: SPALTEN AUSBLENDEN

Um die Zuweisung der Formen nicht zu kompliziert zu machen, soll es auch eine Möglichkeit geben, in einem Template gewisse Spalten auszublenden. Diese sollen ausgeblendet werden, aber trotzdem geladen - und somit später frei wählbar wieder aktivierbar werden.

ANZEIGE IM INTERNET

Wenn die "Zuschauer zuhause" in der Lage sein sollen, die Events live mitzuerleben, sollte es in der Hand des Betreibers liegen, ob und welche Informationen diese bekommen.
Dafür muss eine Übersichtsseite generiert werden, die dann Links auf das Livescoring oder das Informationssystem für Sprecher anzeigt.

ANZEIGE IM WIRELESS LAN

Im lokalen Wireless Lan könnte ein eigener Server stehen, der die Daten erhält. Dazu muss das System natürlich unterscheiden, ob es gerade im Internet oder auf einem lokalen Server arbeitet, da die Arbeitsweise und die Datenbankanbindung unterschiedlich sind. Hier ist die Verzögerung bei der Datenübertragung viel geringer und kann beinahe ignoriert werden.

ANZEIGE ÜBER EIN FERNSEH-SIGNAL

Das Fernsehsignal wird über einen DVI-HDMI Konverter generiert. Das bedeutet, dass es eine Möglichkeit geben muss, eine Anzeige so zu optimieren, dass die Auflösung von 1920x1080px komplett genutzt werden kann.

ANZEIGE AUF MOBILEN GERÄTEN

Die nächste Anzeigevariante: Mobile Geräte.
Während für manche Anzeigen schon gar keine Anpassung mehr erforderlich ist, da viele Geräte schon eine funktionierende Zoommöglichkeiten integriert haben bzw. manche Bildschirme einfach sowieso groß genug sind, ist noch zu untersuchen ob die Funktionalität bei mobilen Geräten leidet.

DOKUMENTATION

Die Dokumentation für dieses Projekt muss bei der technischen Umsetzung genau sein, da das System ein paar Konzepte benutzt, die zwar leicht zu verstehen und zu verändern sind, aber im Code nicht leicht nachvollziehbar sind. Gerade z.B. beim Caching musste Storytelling [122] gegenüber Performance hintangestellt werden. Das wird im Detail bei der praktischen Umsetzung besprochen.

Die Dokumentation dieses Projekts (PHPDoc und JavaScript) ist der Arbeit in CD-Form angehängt.

PRAKTISCHE UMSETZUNG

ÜBERSICHT

Gegeben durch die vielen verschiedenen Anforderungen ist das Konzept nicht auf Anhieb verständlich.

Über ein Zusammenhangs-Diagramm soll ersichtlich werden, wie das System zusammenhängt.

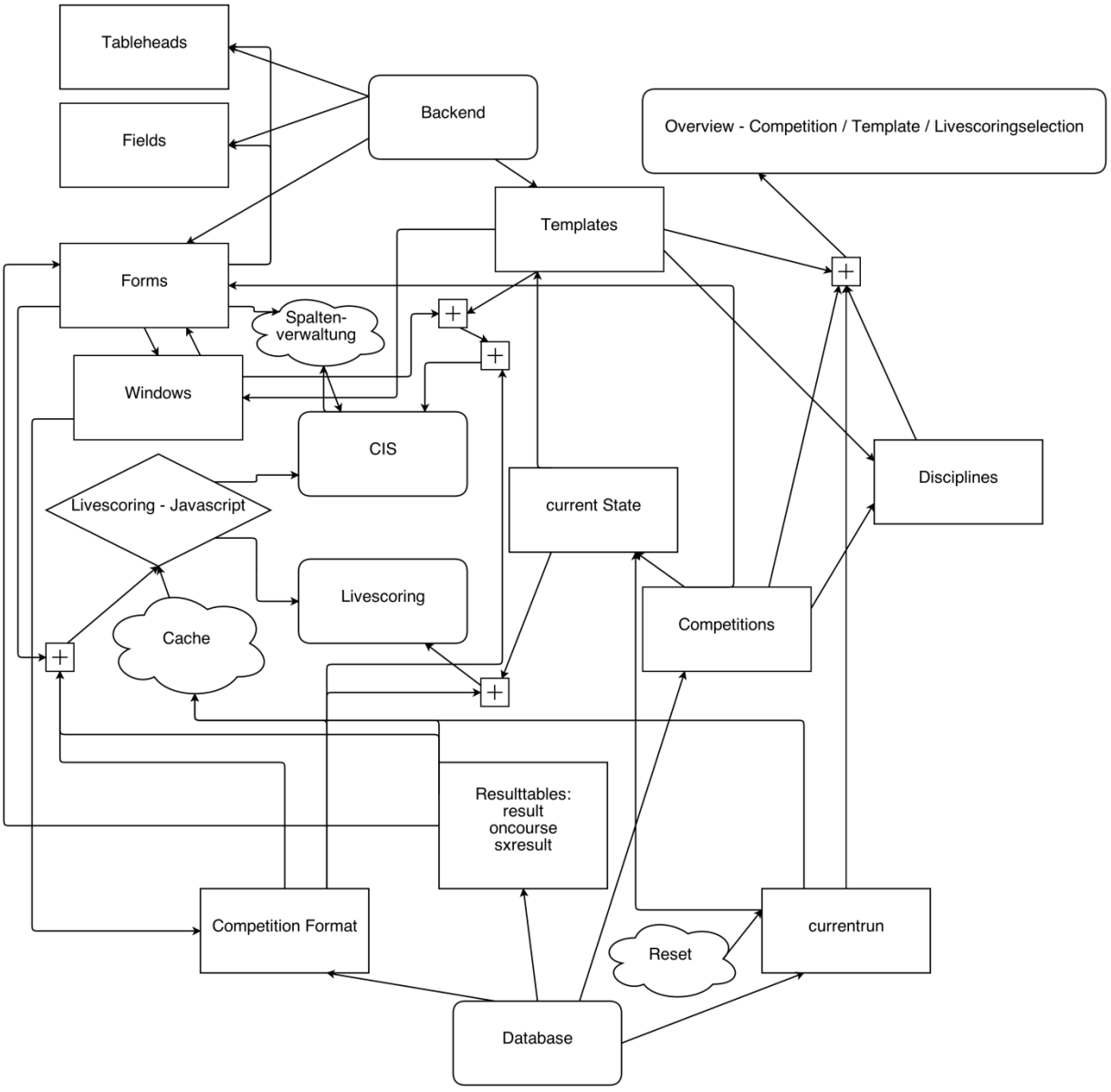


Abbildung 25: Systemübersicht

Da dieses Diagramm noch viele Fragen offen lässt, folgt hier eine genauere Beschreibung:

In der Datenbank (`Database`) gibt es mehrere Tabellen: Für uns wichtig sind die Tabellen `competitionformat`, die Ergebnistabellen `result`, `oncourse` und `sxresult`, sowie die Tabelle `currentrun` und `competition`.

Die Tabelle Competitions enthält die wichtigsten Informationen über jeden einzelnen Wettbewerb: Namen, Austragungsort, Disziplin, Identifikationsnummer, Format und bevorzugte Form für das Livescoring, das CIS separat für die Qualifikation und das Finale.
Die Templates, die pro Disziplin eingestellt sind, sind dann auf der Übersichtsseite für alle Competitions auswählbar. Ebenso kann man bei jeder Competition dort das Livescoring-System aufrufen (das für jede Competition, bis auf das Format und die Form, gleich ist).

Die Tabelle `CompetitionFormat` enthält eine Abbildung sämtlicher Heats, die in einem Wettbewerb dieses Formats vorkommen können. So kann ein Wettbewerb z.B. nur aus der Qualifikation und dem Finale bestehen, aber auch aus mehreren Qualifikationsrennen und Achtel-, Viertel-, Halb- und Finalrennen. Der Name sowie die benötigte Filterung stehen in dieser Tabelle. So sind das Ergebnis und der Cache wie auch die Windows, direkt abhängig davon, ob das Format richtig eingestellt ist.

Die Ergebnistabellen `result`, `oncourse` und `sxresult` sind jeweils etwas verschieden. Die `result`-Tabelle wird z.B. für die Ergebnisse von allen Qualifikationsrennen verwendet, sowie für die Finalrennen von Aerials [123] [124], Moguls [125], Halfpipe [126] und Slopestyle [127]. Hier kommen die Competitors einzeln ins Ziel und bekommen Bewertungen (für Zeit und/oder andere Leistungen).
Die `oncouse`-Tabelle wird im CIS verwendet und enthält ausschließlich die Teilnehmer, die gerade auf der Strecke sind. Sie hat andere Felder als die `result`-Tabelle, da z.B. ja noch kein endgültiger Rank angezeigt werden kann.
Die `sxresult`-Tabelle ist eine Art Mix zwischen `result` und `oncourse`. Die Einträge werden regelmäßig aktualisiert - je nach Verbindung vor Ort nach jedem Durchgang oder auch nach jedem Zwischenergebnis (Split). Ebenso gibt es in der `sxresult`-Tabelle Einträge für den Track sowie die verwendete Trikot-Farben.
Diese Tabellen werden für jede Form ausgewählt und beeinflussen dort dann die

Auswahlmöglichkeit im Field-Feld. Über die aufsteigend und einzigartig vergebenen Ergebnis-Identifikationsnummer [128] (`rid`) kann das Caching-System in der `result` und `sxresult`-Tabelle erkennen, bei welchen Zeilen neue Ergebnisse eingetragen worden sind.

In der currentrun-Tabelle steht für jede Competition, welchen Status sie gerade hat. Das ist das gerade aktive Geschlecht, die Filterinformation für den aktuellen Run (mit dem dann wieder ein Rückschluss über die CompetitionFormat auf den aktuellen Heat möglich ist) und eine Nachricht, die gegebenenfalls im CIS eingeblendet wird. Das clientseitige System, das den current State überwacht, schickt periodisch eine Abfrage an diese Tabelle. Bei einer Änderung wird das System aktiv und ändert den Status des CIS/Livescoring entsprechend. Beim CIS wird auch je nach Status das Template umgetauscht, falls beim Template eine Unterscheidung zwischen Qualifikation und Finalrennen eingebaut ist. Eine wichtige Funktion dieses Systems ist auch der Reset der Benutzerdaten.

Das Caching-System wird mit allen Daten gefüttert, die an das Livescoring oder CIS geliefert werden und liefert diese an alle gleiche nachfolgenden Anfragen aus. Es ist abhängig vom Format, vom System (Livescoring/Cis) sowie vom Fenster und vom Current-Run System (reset). Diese Elemente können alle die Form ändern.

Im Backend werden alle übergreifenden Funktionen definiert. Z.B. die Tableheads, Felder und Aufteilung von Formen, die Namen und Zugehörigkeit[9] von Templates, die Fenster pro Template, sowie für die Fenster die bevorzugte Form (die die Form, die beim Wettbewerb gesetzt ist, überschreiben kann), Größe und Position. Die Reihenfolge und Anwendung von diesen Anzeigevarianten wird später genauer beschrieben.

[9] zu Disziplinen

Um den Vorgang leichter verständlich zu machen, soll er hier visualisiert werden:

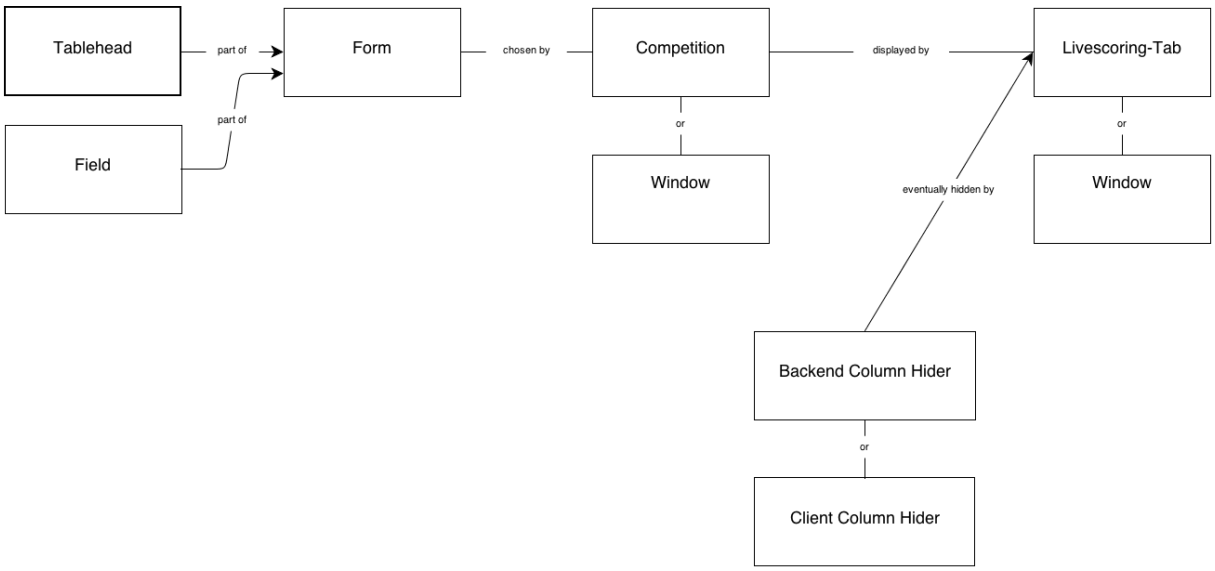


Abbildung 26: Weg eines Tabellenkopfes zum Client

Das CIS vereint alle Funktionen: Es zeigt ein vorher definiertes `Template` einer bestimmten `Competition` an. Von diesem `Template` zeigt es alle `Windows` in der eingestellten oder überschriebenen `Form` an. Je nach Einstellung blendet es über die `Spaltenverwaltung` der `Windows` die von der `Form` vorgegebenen Spalten aus. Es ladet über das `Livescoring-Javascript` periodisch abwechselnd für jedes `Window` die inkrementell geänderten Ergebnisse - meistens vom `Cache`, da nur der erste Zugriff direkt auf die `Result-Tables` geht. Das `Current State` - System überwacht mit Hilfe vom `Competition Format` ob der richtige Run angezeigt wird. Es wechselt gegebenenfalls das Template aus bzw. regt das `Livescoring-Javascript` an, Ergebnisse auszublenden, damit das aktuelle Ergebnis sichtbar ist. Bei einem `Reset` werden die Ergebnisse in allen `Windows` zurückgesetzt und neu geladen.

Das Livescoring ist ähnlich wie ein abgespecktes CIS. Wenn man die Tabs als Element den Windows gleichstellt, und sie modifizert, sodass nur ein Element auf einmal aktiv werden kann, ist das System schon fast zum Livescoring umgewandelt. Nur muss das Livescoring keine Ergebnisse ausblenden sondern im Gegensatz immer alle Ergebnisse sichtbar halten.

CACHING

Der rechenaufwendigste Teil der Generierung der Ergebnisse war die Generierung der Ergebnislisten, mit den Namen und den Herkunftsländern der Teilnehmer. Ebenso war hier der höchste Anteil an Ergebnissen, die für mehrere Benutzer des Systems gleich sind.

Die Quintessenz des Systems sind die statischen Dateien, die vom Server ausgeliefert werden. Das bedeutet, dass im Idealfall für ein neues Ergebnis nur einmal auf die Datenbank zugegriffen werden muss. Doch auch schon dieser Idealfall zeigt sich gleich als unerfüllbar - muss es doch für verschiedene Formen vom selben Ergebnis jeweils eine andere Datei geben. Da das Livescoring und das CIS verschiedene Formen haben können und auch im CIS in zwei verschiedenen Windows verschiedene Formen verwendet werden können, dürfen diese nicht vermischt werden.

Um dieses Thema verständlich zu machen, soll hier als erstes die Erstellung einer Ergebnisdatei visualisiert werden:

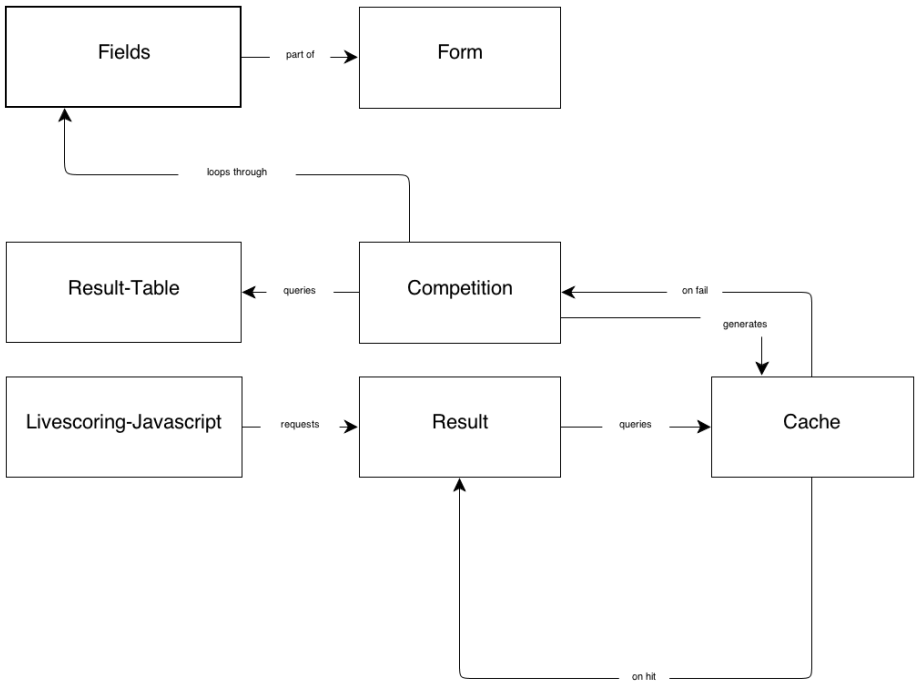


Abbildung 27: Erstellung von einer Ergebnisdatei

Das Livescoring-JavaScript fordert periodisch eine Ergebnisdatei an - je nach Verbindungsgeschwindigkeit stellt das System die Pausen zwischen den Abfragen höher oder niedriger. Das stellt eine zweite Instanz der Serverbelastungsreduktion dar. Bei längerer Abfragedauer wird auf eine Überlastung des Systems geschlossen und deswegen die Frequenz der Anfragen reduziert.

Die angefragte `Competition` überprüft in einem `JOIN` aus der gegebenen Ergebnistabelle und dem Cache, ob eine neue Datei verfügbar ist bzw. erstellt werden muss.
Dabei müssen mehrere Faktoren in Betracht gezogen werden - schließlich gibt es verschiedene Kombinationen von Formen und Ergebnissen.
Eine grundlegende Unterscheidung ist, ob es sich um eine Startliste oder eine Ergebnisliste handelt.

Bei einer Startliste gibt es die folgenden Unterscheidungspunkte:

- Erstellungszeit
  Bei der Startlistenerstellung wird die ID des zuletzt eingefügten Starters keine verlässliche Auskunft darüber geben, ob eine neue Datei erstellt werden soll. Gleichzeitig werden sich eher selten Dinge ändern, die ein Neuladen der Startliste nötig macht. Daher soll periodisch nach einer gewissen Zeit eine neue Startliste erstellt werden.
  Je nach Art der Startliste (Insgesamt, verbleibende Starter, nächster Starter) müsste nicht nur eine Liste geschickt werden welche Starter hinzugekommen sind, sondern auch eine Liste, welche Starter nicht mehr anzuzeigen sind. Da der Overhead davon viel zu groß ist, wird bei solchen Startlisten die gesamte Liste der momentan aktiven Starter gezeigt.
- Form
  Welche Form soll die Startliste haben? Eine gewöhnliche Startliste ist fix definiert, allerdings kann bei den anderen Arten der Startliste auch die Form frei definiert werden.
- Fenstertyp
  Auch wenn die Form gleich ist, gibt es für verschiedene Fenstertypen (Art der Startliste) verschiedene Anforderungen. So wird eine Gesamtstartliste nicht das gleiche anzeigen wie die Liste, die die verbleibenden Starter anzeigt.

- Competition
  Natürlich muss einfließen, für welche Competition dieses Ergebnis ist
- Format-ID
  Welcher Run/Heat ist gerade aktiv? Die Startlisten z.B. für Qualifikation und Finalrennen müssen unterscheidbar sein.
- Geschlecht
  Welches Geschlecht ist gerade aktiv?

Mit diesen Informationen kann eine Startlistendatei generiert werden, die dann erst nach vorgegebener Zeit ausgetauscht wird.

Für die Erstellung einer Ergebnisliste müssen folgende Punkte beachtet werden:

- Form
  Die Form der Ergebnisliste - vorgegeben vom Request
- Fenstertyp
  Auch bei den Ergebnislisten können die Fenster verschiedene Anforderungen haben
- Competition
- Format-ID
- Geschlecht
- Last Result - ID
  Bei den Ergebnislisten kann bei manchen Fenstertypen und beim Livescoring die meiste Bandbreite gespart werden, indem nur Unterschiede zwischen den Ergebnislisten zum Client geschickt werden.

Wenn mit diesen Daten ein Eintrag vorhanden ist, wird der Dateiname dieses Eintrags generiert und zurückgeschickt. Wenn nicht, wird durch die Form geloopt und die zugehörigen Felder ausgelesen, wie im folgenden Diagramm erklärt.

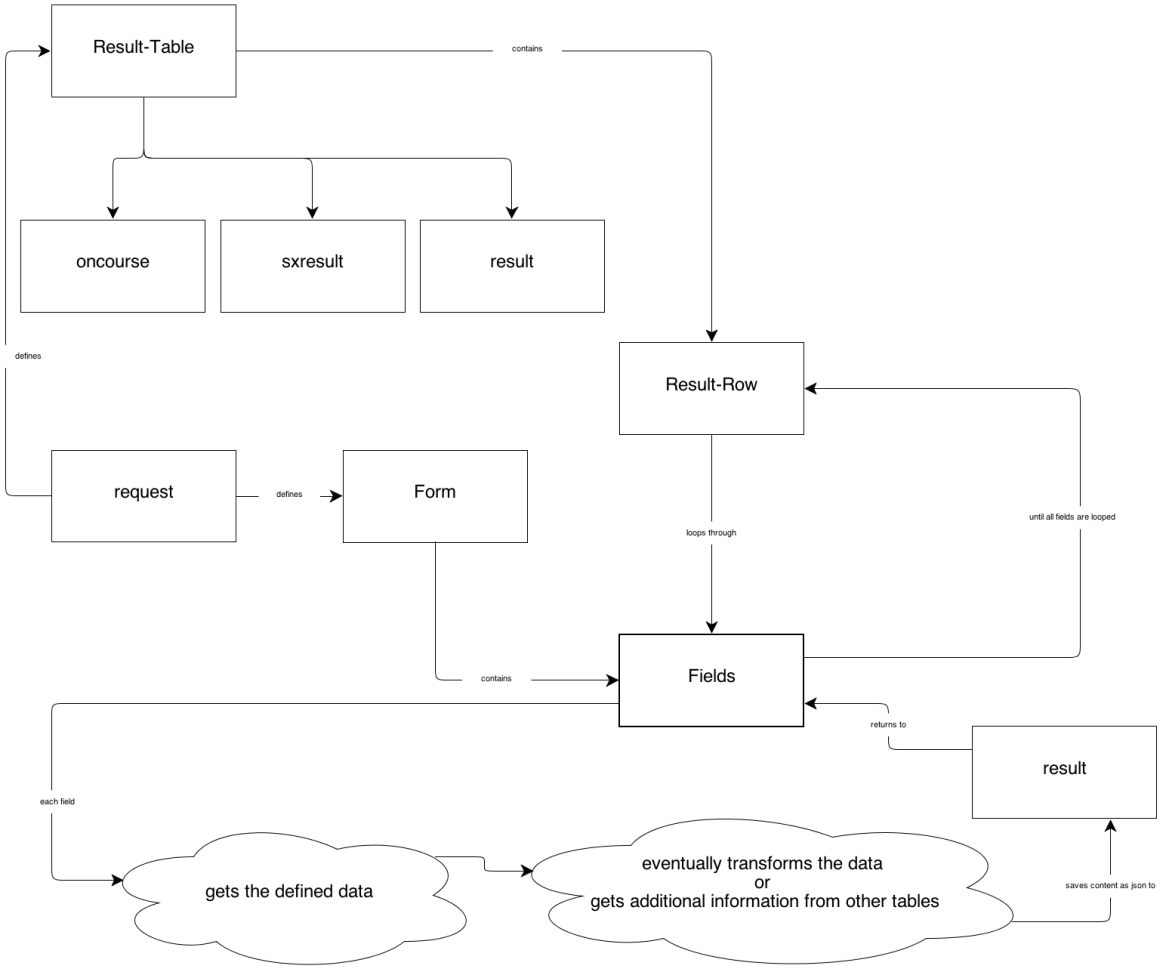


Abbildung 28: Erstellung der Result-Files

JAVASCRIPT-CACHING

LIVETICKER

Als erstes soll der Ticker selbst beschrieben werden, der periodisch Abfragen an den Server schickt und nach neuen Informationen fragt. Um diese Beschreibung so greifbar wie möglich zu machen, wird der Code direkt eingebunden und erklärt.

Die ersten Zeilen initiieren den Liveticker und geben Standardwerte an. So wird zum Beispiel ein Platzhalter definiert, der den Timeout-Aufruf der Funktion beinhaltet, danach die Definition, ob es sich um das CIS handelt. Dann wird die ursprüngliche Zeit eingespeichert, mit der der Liveticker aktualisiert. Die Zeiten danach geben in Millisekunden an, was die Ober- und Untergrenze für die Zeit zwischen zwei Abfragen an den Server ist. Die letzte Zeile von diesem Block setzt ein Intervall, das den Liveticker alle 5 Sekunden neustartet, falls er hängengeblieben ist. Falls der Liveticker funktioniert, würde dieses Intervall automatisch gecancelt werden.

```
//prevent 2 tickers at the same time
liveticker.timeout = 0;
liveticker.cis = false;
liveticker.original_timeout = 0;
liveticker.min_timeout = 500;
liveticker.max_timeout = 3000;
liveticker.timeout_ms = 2000;
setInterval(function (){liveticker.timeout = setTimeout(liveticker,
4500);}, 5000); //restart all 5 sec, but only if the ticker dies
```

Codebeispiel 12: Initialisierung des Live-Tickers

Wenn die Abfrage langsamer zurückgekommen ist als die momentane Abfragegeschwindigkeit ist, soll der Abstand verlängert werden, andernfalls verlängert. Die folgenden beiden Funktionen erledigen das.

```
liveticker.increaseTimeout = function()
{
  if(liveticker.original_timeout == 0) liveticker.original_timeout =
liveticker.timeout_ms;
  if(liveticker.timeout_ms < liveticker.max_timeout)
liveticker.timeout_ms += liveticker.timeout_ms / 10;
};
```

```
liveticker.decreaseTimeout = function()
{
  if(liveticker.original_timeout == 0) liveticker.original_timeout =
liveticker.timeout_ms;
  if(liveticker.timeout_ms > liveticker.min_timeout)
liveticker.timeout_ms -= liveticker.timeout_ms / 10;
};
```

Codebeispiel 13: Liveticker - Geschwindigkeitsanpassung

Die Funktion stoppt als erstes simultan ablaufende Anfragen wie z.B. den Ticker-Neustart oder eventuell parallel laufende Instanzen. Falls das System ein CIS-System ist, wird das zu aktualisierende Fenster gewechselt und weitere Informationen an den Abfragestring angehängt.

```
function liveticker()
{
  clearTimeout(liveticker.timeout);
  liveticker.timeout = 0;
  if(liveticker.result !== undefined) liveticker.result.abort();
  if(display.is_cis && !display.is_livescoring)
display.changeWindow();

  var cisinfo = "";
  if(display.is_cis) cisinfo = "&cis=1";
  if(display.is_cis) cisinfo += "&type="+display.type;
  if(display.is_cis && display.competition_format_form > 0) cisinfo +=
"&cff="+display.competition_format_form;
  display.starttime = display.getTime();
```

Nun wird die Abfrage an den Server geschickt. Dabei wird gemessen, wie lang die Abfrage braucht, um die Frequenz anzupassen.
Die Ergebnisdaten werden in die verarbeitende Funktion loadResultData geschickt, wo sie den richtigen Tabellen zugeordnet werden.

```
  liveticker.result =
$.getJSON('live/getData.php?c='+display.getCompetitionId()+'&cfi='+dis
play.getCurrentRunFormatId()+'&startlist='+display.isStartList()+'&gen
der='+display.getGender()+"&last=" + display.getLast()+cisinfo,
function(data)
  {
    if(display.getTime() - display.starttime > liveticker.timeout_ms)
liveticker.increaseTimeout();
    else liveticker.decreaseTimeout();
    loadResultData(data);
    liveticker.timeout = setTimeout(liveticker,
liveticker.timeout_ms);
  });
}
```

Codebeispiel 14: Der Liveticker

Der Reset wird vom überwachenden System aufgerufen, das den Status des Livescoring überwacht. Falls sich die Version der Datenbank von der Version des Livescoring unterscheidet, wird der Reset getriggert.

Falls es sich um ein CIS handelt, werden alle Fenster, die momentan verwaltet werden, zurückgesetzt. Beim Livescoring handelt es sich nur um eine Tabelle, und hier wird noch eine Information an den Benutzer ausgegeben.

```
liveticker.reset = function()

{
  if(display.is_cis)
  {
    $.each(display.windows, function(index, value)
    {
      display.windows[index].last = 0;
      $('#'+display.windows[index]["id"] + ' table').find("tbody
tr").remove();
    });
    display.last = 0;
    display.getTable().find("tbody tr").remove();
  }
  if(display.is_livescoring)
  {
    display.resetLast();
    var table = display.getTable();
    var amount_of_cols_to_span = table.find('thead tr th').length;
    table.find('tbody tr').remove();
    table.trigger("update");
    table.append("<tr><td colspan=\"" + amount_of_cols_to_span +
"\">Loading Data...</td></tr>");
  }

};
```

Codebeispiel 15: Javascript-Seitiges Caching

STATUS-ÜBERWACHUNG

Da die Statusüberwachung einen völlig neuen Aspekt bringt, soll hier die gesamte Funktionalität dargestellt werden. Die Überwachung beginnt damit, dass serverseitig beim Aufruf der Initialstatus gesetzt wird.
Auf die Abfragen, die zum Bekommen dieser Daten nötig sind, wird in dieser Darstellung bewusst verzichtet, da es sich hierbei um Rückgabefunktionen von Membervariablen bzw. um simple SQL-Queries handelt.

Beim Initialstatus werden dem System Daten wie der Name, der momentane Status des Rennens und die aktuelle Version gesetzt. Im CIS wird auch eine niedrigere Aktualisierungsgeschwindigkeit gesetzt, da das CIS wohl am ehesten für lokale Sprecher sinnvoll ist.

```
  display.competitionname = '<?php echo $competition-
>getCompetitionName();?>';
  display.current_run_format_id = <?php echo $competition->
getCompetitionFormatCurrent();?>;
  display.gender = '<?php echo $competition->getCurrentGender();?>';
  display.current_run_form_id = <?php echo $competition->
getCompetitionCisForm();?>;
  display.quali_run_form_id = <?php echo $competition-
>getCompetitionCisForm(0);?>;
  display.finals_run_form_id = <?php echo $competition->
getCompetitionCisForm(1);?>;
  display.competition = <?php echo $competitionId?>;
  display.result_revision = <?php echo $competition-
>getResultRevision();?>;
  liveticker.timeout_ms = 250;
  liveticker.min_timeout = 150;
  liveticker.max_timeout = 800;
  display.is_cis = true;
  display.layer_1_template = <?php echo $final_template_for;?>;
  display.layer_0_template = <?php echo $qualification_template;?>;
  display.is_final_view = display.layer_0_template > 0;
  display.layer = display.layer_0_template > 0 ? 1 : 0;
```

Codebeispiel 16: Initialisierung des Systems

Die `checkRun` - Funktion ist ähnlich aufgebaut wie der Liveticker. Auch diese Funktion verhindert als erstes, dass parallele Instanzen dieser Anfrage laufen und hängt wenn nötig die Information an, ob es sich um das CIS handelt.

Anders als beim Livescoring löst diese Information hier nur aus, dass eine eventuell vorhandene Nachricht an den Client ausgegeben wird. Eine Anpassung der Frequenz ist hier aufgrund der sowieso sehr hohen Abstände nicht nötig.

```
display.checkRun = function()
{
  clearTimeout(display.check_run_timeout);
  display.check_run_timeout = 0;
  if(display.checkrun_request !== undefined)
display.checkrun_request.abort();

  var cisinfo = "";
  if(display.is_cis) cisinfo = "&cis=1";

  display.checkrun_request =
$.getJSON('live/getState.php?c='+display.getCompetitionId()+cisinfo,
function(data)
  {
    display.checkRunData(data);
    display.check_run_timeout = setTimeout(display.checkRun, 10000);
  });
};
```

Codebeispiel 17: Überprüfung des momentanen Status der Competition

In der folgenden Funktion werden die Daten verarbeitet. Falls keine Daten vorhanden sind, kann das z.B. bedeuten, dass die Veranstaltung noch in keinen gültigen State gesetzt wurde.
Diese Funktion erklärt sich selbst beim Lesen - wenn ein Wert anders als der abgespeicherte Wert ist, werden die entsprechenden Funktionen gestartet.

```
display.checkRunData = function(data)
{
  if(typeof data != "object") return;
  if(data["Layer"] == null) return;

  /**
   * revision changed? reload anyway!
   */
  if(data["ResultRevision"] != display.result_revision)
  {
    display.switchRevision(data["ResultRevision"]);
  }

  /**
   * livescoring: don't want to track the current run - just uncheck
this box
   */
  if(display.is_livescoring && !$('#advance').is(":checked")) return;
```

Als kleine Besonderheit ist hier noch die Nachrichtenübertragung genauer zu betrachten. Die Nachricht wird jedes Mal übertragen, falls sie gesetzt ist. Das bedeutet, dass im Normalfall (keine Nachricht gesetzt) keine Bandbreite verbraucht wird. Sollte die Nachricht jedoch länger stehen, wird sie bei jedem Aufruf mitgeschickt.

```
  if(data["message"] != display.message)
  {
    display.switchMessage(data["message"]);
  }
  if(data["Layer"] != display.layer)
  {
    display.switchLayer(data["Layer"]);
  }
  if(data["currentRun"] != display.current_run)
  {
    display.switchRun(data["currentRun"]);
  }
  if(data["currentFormatId"] != display.current_run_format_id)
  {
    display.switchFormat(data["currentFormatId"],
data["currentFormatName"]);
  }
  if(data["Gender"] != display.gender)
  {
    display.switchGender(data["Gender"]);
  }
  if(display.is_cis)
  {
    var cistitle = display.competitionname;
    if(display.gender != "") cistitle += " | " + display.gender;
    if(display.current_format_name != "") cistitle += " | " +
display.current_format_name;
    $('#cistitle').text(cistitle);
  }
};
```

Codebeispiel 18: Überprüfung des aktuellen Status, Fortsetzung

In dieser Funktion wird das aktuell aktive Fenster ausgetauscht. In das derzeit aktive Fenster wird der letzte Competitor gespeichert und anschließend das aktive Fenster hintan gereiht. Dann werden die gültigen Werte für das aktive Fenster in die Anzeige eingespeichert. Die Anzeige funktioniert als Wrapper bzw. Adapter damit für die Funktion die die Ergebnisse verwaltet alle Fenster sowie CIS und Livescoring gleichwertig sind.

```
display.changeWindow = function()
{
    if(typeof display.actual_window == "object")
    {
      display.actual_window["last"] = display.last;
      display.windows.push(display.actual_window);
    }
    display.actual_window = display.windows.shift();

    display.table = $('#'+display.actual_window["id"] + ' table');
    display.last = display.actual_window["last"];
    display.type = display.actual_window["type"];
    display.competition_id = display.actual_window["competition_id"];
    display.competition_format_form =
display.actual_window["competition_format_form"];
};
```

Codebeispiel 19: CIS: Fenster-wechsel für die Aktualisierung

Damit die Fenster in dieser Liste stehen können, muss es eine Funktion geben, die diese in das Array schreibt. Das erledigt die Funktion `addWindow`. Eine erste Besonderheit ist die Spezialbehandlung für den `WINDOWTYPE_CURRENT_RUNNER`. Damit dieser Fenstertyp korrekt angezeigt werden kann, muss das System von dem Server die aktuelle Zeit abfragen. Das erledigt ein eigener Request, der eine PHP-Datei vom Server abfragt, der nur die aktuelle Zeit ausgibt. Mit der ermittelten Zeit kann das System dann den Zeitunterschied zwischen Server und Client errechnen. Der Unterschied wird zur Serverzeit addiert (bzw. abgezogen) und damit die vergangene Zeit errechnet.
Die restlichen übergebenen Daten werden in ein Objekt gespeichert, das dem Windows-Array angehängt wird.

```
display.addWindow = function(type, id, competition_id,
competition_format_form)
{
  if(type == display.WINDOWTYPE_CURRENT_RUNNER)
  {
    display.updateTime();
  }
  if(typeof display.windows == 'undefined') display.windows = [];
  display.windows[display.windows.length] =  {
    "type" : type,
    "competition_id" : competition_id,
    "last" : 0,
    "alwaysvisible" : [],
    "competition_format_form" : competition_format_form,
    "id" : id
  };
};
```

Codebeispiel 20: Fenster-Container hinzufügen

Das Format bestimmt, was der aktive Run ist. Bezug ist hier die `competition_format` -Tabelle bzw. die ID der gerade aktiven Reihe. Im Livescoring soll diese Änderung keine Auswirkungen haben, da es im Livescoring keine Rückschlussmöglichkeit von der Format-ID auf den aktiven Tab gibt und auch keine Überschrift aktualisiert werden muss, in der der Name des aktiven Runs steht.

```
display.switchFormat = function(format, formatname)
{
  if(display.is_livescoring) return;
  if(!display.is_cis)
  {
    return;
  }
  display.current_run_format_id = format;
  display.current_format_name = formatname;
  if(display.is_final_view) return;
  liveticker.reset();
};
```

Codebeispiel 21: Format des aktuellen Rennens ändern

Die Nachricht, die in der `competition`-Tabelle steht, wird - wie bereits beschrieben - immer mitgeschickt, wenn sie aktiv ist. Falls die Nachricht also leer ist, wird nach der Entfernung der Nachricht einfach keine neue erstellt. Da die Entfernung und Neuerstellung nicht wesentlich länger dauert als ein Vergleich dieser gespeicherten Werte, wird dieser weniger fehleranfällige Weg gegangen.

```
display.switchMessage = function(message)
{
  if(display.is_livescoring) return;

  display.message = message;

  $('#info').remove();

  if(trim(message) == "")
  {
    return;
  }
  $('body').append($('<div id="info">'+message+'</div>'));
};
```

Codebeispiel 22: Aktuelle Nachricht anpassen

Der momentan aktive Run benötigt keine Nebenaktionen - dieser Wert ist nur für die Abfrage vom Liveticker wichtig.

```
display.switchRun = function(run)
{
  display.current_run = run;
};
```

Codebeispiel 23: Aktuellen Run setzen

Der Layer, der bestimmt ob ein Wettbewerb in der Qualifikation oder im Finale steckt, ist umso wichtiger. Im Livescoring wird der Tab mit dem aktiven Run angeklickt, im CIS wird je nach Verfügbarkeit und Notwendigkeit auf ein anderes Template gewechselt, was ein Neuladen der Seite benötigt.

```
display.switchLayer = function(layer)
{
  display.layer = layer;
  if(display.is_livescoring)
  {

    if(display.brackets_active && display.layer > 0)
    {
      $(".result:last").click();
    }
    else $('#result'+layer).click();
  }
  if(!display.is_cis) return;
  if(display.is_livescoring) return;
  if(display.layer >= 1 && display.layer_1_template > 0)
  {

window.location.replace("cis.php?c="+display.competition+"&template="+
display.layer_1_template);
  }
  if(display.layer == 0 && display.layer_0_template > 0)
  {

window.location.replace("cis.php?c="+display.competition+"&template="+
display.layer_0_template);
  }
  if(display.current_run_form_id != display.quali_run_form_id)
    location.reload();
};
```

Codebeispiel 24: Aktuellen Layer des Systems ändern

Für den Fall, dass sich die Version geändert haben sollte, also ein Reset geschickt wurde, gibt es zwei Überlegungen. Falls man davon ausgeht, dass sich auch andere Dinge neben den Ergebnissen selbst geändert haben (z.B. das Template), kommt man nicht umhin, die gesamte Seite neu zu laden. Auskommentiert sieht man die zweite Lösung - falls es nur um Daten geht, würde es reichen den Liveticker zu resetten und die neue Versionsnummer einzuspeichern.

```
display.switchRevision = function(revision)
{
  location.reload();
//  liveticker.reset();
//  display.result_revision = revision;
};
```

Codebeispiel 25: Revision des Systems

Ein häufiges Szenario ist das Wechseln zwischen Männern und Frauen, also Men und Ladies, um in der beim FIS üblichen Bezeichnung zu bleiben.
Falls das CIS gerade aktiv ist, muss nur das Geschlecht gesetzt und die Daten entfernt werden. Die Überschrift wird durch `checkRunData` angepasst, sonst ändert sich im CIS (bis auf die Ergebnisse) nichts.
Im Livescoring muss die verfügbare Checkbox gesetzt werden. Die Funktion die beim Wechseln gestartet wird gleich aktiviert.

```
display.switchGender = function(gender)
{
  display.gender = gender;
  if(display.is_cis)
  {
    liveticker.reset();
  }
  if(display.is_livescoring)
  {
    if($("input[name='gender']:checked").val() == gender) return;
    else
    {
      $("input[name='gender']").each(function(index, value){
        if($(value).val() == gender) $(value).click();
      });
    }
  }

};
```

Codebeispiel 26: Wechsel zwischen den Geschlechtern

SERVERSEITIGES CACHING

LIVE/GETDATA.PHP

Diese Datei wird während eines Rennens unzählige Male aufgerufen. Alle vom Javascript übergebenen `GET`-Werte werden ausgelesen und in eine sichere Form für Datenbanken gebracht. Die Funktion `getInt` parst einen Wert aus dem `GET` - Array in eine Zahl. Falls es keine Zahl ist, wird 0 zurückgegeben.
Die Competition wird über eine statische Funktion als Factory [129] generiert. Das bietet den Vorteil, dass die `Competition`-Klasse selbst wissen muss, welche Klasse[10] sie für welche Competition ausgibt.
Für den verwendeten String (Gender) kann hier ein einfaches Whitelisting [130] verwendet werden, was die höchste Sicherheit gegen Escape-Sequenzen bietet.

```
<?php
include('../include/include_general.php');

$competitionId = getInt("c");
$competition = competition::generateCompetition($competitionId);

//male / female (L = Ladies)
$gender = $_GET["gender"] == "M" ? "M" : "L";
$competition->setGender($gender);
//qualification / final
$competition_format_id = getInt("cfi");
$competition_format_form = getInt("cff");
if($competition_format_form > 0)
  $competition->setCompetitionCisForm($competition_format_form);
if($competition_format_form > 0)
  $competition->setCompetitionCisFinalForm($competition_format_form);
$competition->setActualRun($competition_format_id);
$competition->setLastCachedResultId(getInt("last"));
if(getInt("cis") > 0)
  $competition->setIsCis(true);
if(getInt("type") > 0)
  $competition->setCISWindowType(getInt("type"));
```

Die ausgewählte Competition-Klasse kann nun eine bereits erstellte Ergebnisdatei zurückgeben oder selbst eine erstellen. Falls keine neuen Daten verfügbar sind, würde `"null.json"` zurückgegeben werden. Damit hierfür keine neue Verbindung zum Server aufgebaut werden muss, gibt die Datei hier selbst ein leeres JSON-Objekt zurück.

[10] Competition ist die Basisklasse für die sechs Unterklassen

```
if($_GET["startlist"] == "true")
  $result_file = $competition -> getStartListFile();
else
  $result_file = $competition -> getResultFile();

header("Cache-Control: no-cache, must-revalidate");
header("Expires: Sat, 26 Jul 1997 05:00:00 GMT");
if($result_file == "null.json") die("[]");
header("Location: ../cache/$result_file", true, 302);
exit;
```

Codebeispiel 27: Serverseitige Request-Datei

CACHE-ABFRAGE

Wie bereits erwähnt, musste beim Caching eindeutige Lesbarkeit für die Code Performance weichen. Im Speziellen bedeutet das, dass hier eine SQL-Query, die 2 Tabellen vereint, die gesamte Überprüfung auf neue Daten erledigt. Das reduziert die MySQL-Abfragen pro Cache-Abfrage auf 2.

Dieser zentrale Aufruf soll hier detailliert beschrieben werden. Als erstes wird die abzufragende Tabelle definiert. Die normalerweise verwendeten Filter werden gefüllt.

```
  private function existingResultFile()
  {
    $table = "result";
    $additional_filter = " AND a.Gender='" . $this->getGender() . "'
AND a.Layer='" . $this->getLayer() . "' AND (R_OrderBy = '" . $this-
>getRun1OrderBy() . "' OR R_OrderBy='" . $this->getRun2OrderBy() .
"')";
```

Für den Anzeigemodus "Brackets" benötigen wir alle verfügbaren Daten. In diesem Modus befinden sich in einem Fenster oder Tab mehrere Tabellen, markiert nach ihrem Layer und ihrem Filter (`R_OrderBy`). Die Funktion `loadResultData` verarbeitet jedes Ergebnis einzeln und tragt es in die entsprechenden Tabellen ein. Über das `LIMIT 1` wird nur die aktuellste Zeile abgefragt. Da ein 1:n Verhältnis zwischen dem Erstellen des Cache und dem Abfragen danach ist, ist die Geschwindigkeit beim Abfragen des Caches ungleich wichtiger als die Geschwindigkeit beim Erstellen[11].

[11] Die Ergebnis-Abfrage könnte durch diese Query, mit ein paar Umbauten, ersetzt werden und damit eine Abfrage weniger ermöglichen.

```
    if($this->getCISWindowType() == WINDOWTYPE_BRACKETS)
    {
      $table = "sxresult";
      $additional_filter = "";
    }

    $res = db::do_sql("SELECT rid as last, actual_result_id FROM
$table a LEFT JOIN cache c ON (c.actual_result_id = rid AND c.cid='" .
$this->getId() . "' AND c.competition_format_id='" . $this-
>getActualRun(). "' AND c.cis_window_type='" . $this-
>getCISWindowType() . "' AND c.gender = a.gender AND
c.last_result_id='" . $this->getLastCachedResultId() . "') WHERE
a.CompetitionId = '" . $this->getCompetitionStringId() . "'
$additional_filter ORDER BY rid DESC LIMIT 1");

    $row = db::fetch_assoc($res);

    $actual_latest_result_id = $row["last"];

    $this->setActualResultId($actual_latest_result_id);
```

Das Verhalten von der Abfrage, ob eine neue Startdatei verfügbar ist, ist nur geringfügig anders als die Abfrage nach neuen Ergebnissen. Da dabei aber zwei völlig andere Tabellen betroffen sind und vor allem die Logik verschieden ist, gibt es dafür eine eigene Funktion. In dieser wird z.B. nicht die aktuellste Ergebnis-ID gespeichert, sondern die Zeit der Erstellung der Startdatei. Bei einer Startliste kann es nicht keine neuen Ergebnisse geben.

Die nächste Fallunterscheidung überprüft, ob der Cache mit der letzten Ergebnis-ID (die vom Request übermittelt wurde) übereinstimmt.

```
    if ($this->getActualResultId() == $this->getLastCachedResultId()
&& $this->getActualResultId() > 0) return "null.json";
```

Wenn keine Ergebnisse gefunden wurden, ist der Cache leer oder ein Ergebnis neuer als der Cache.

```
    /**
     * no result - nothing cached
     */
    if (db::num_rows($res) <= 0 || $row["actual_result_id"] == NULL)
      return "";
```

Nach der Abfrage, ob eventuell gar kein Ergebnis vorhanden ist bzw. die Ergebnis-ID NULL ist, muss der Vergleich nochmal getätigt werden. Das ist ein seltener Einzelfall.

```
    /**
     * nothing new, go away
     */
    if ($this->getActualResultId() == $this->getLastCachedResultId()
|| $this->getActualResultId() == 0)
      return "null.json";
```

Wenn etwas gefunden wurde, kann der Dateiname berechnet und zurückgegeben werden.

```
    /**
     * return cached file
     */
    return $this->getResultFilename();

  }
```

Codebeispiel 28: Serverseitiges Caching

FORMEN

AUSGABE

Um die größtmögliche Flexibilität gleichzeitig mit hoher Performance zu ermöglichen, wurde folgende Lösung angewendet:

Die verwendete Form wird zuerst abgefragt, danach die Ergebnistabelle. Da die Form pro Ergebnis einmal benötigt wird, wird das Form-Ergebnis einfach zurückgesetzt [131] und neu eingelesen. Dadurch reduzieren sich die Datenbankabfragen für das Auslesen der Ergebnistabellen selbst (abgesehen von den Namen und der Nation der Competitors) auf zwei Stück.

Um die Balance zwischen Single Responsibilty [132] und in Hinblick auf spätere Maintenance und Informationszuständigkeit zu halten, wurde diese Funktion in der Klasse Competition behalten. Diese Funktion ist über die Evolution des Systems für alle Competitions gleich geworden. Möglich z.B. wäre auch eine Auslagerung an die Form selbst gewesen, oder eine eigene Klasse, die die Requests verwaltet. Manchmal sind solche Konstrukte allerdings dann doch schwerer zu durchschauen als eine Klasse, die 1-2 Funktionen zu viel hat.

Die Klasse bestimmt also, welche Tabelle mit welchen Faktoren ausgelesen wird. Zu den Faktoren zählt zum Beispiel, welche Ergebnisse neu sind, da die Änderungen nur inkrementell ausgegeben werden (siehe Caching). Diese Tabelle wird dann Zeile für Zeile bearbeitet. Für jede Zeile wird ein Durchgang durch die Form gemacht, wodurch dann nur die benötigten Informationen an das Frontend geschrieben werden, wie unter Caching genauer beschrieben.

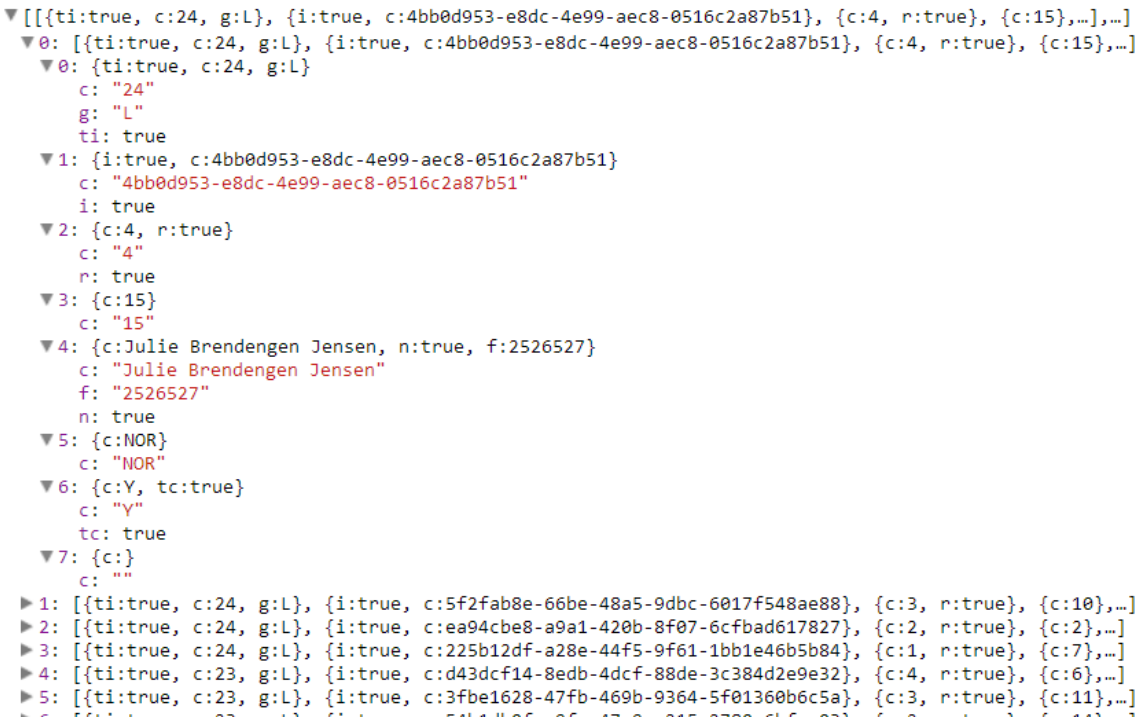


Abbildung 29: Ausschnitt von JSON-Daten

STRUKTUR

Die Form bestimmt, was von der Backend-Seite auf die Frontend-Seite übermittelt wird. Es wird ein Array pro Teilnehmer übergeben, mit einem assoziativen Array (Ein Objekt in JSON [133]) für jedes Feld. Jedes Feld ist seinerseits dann wieder ein assoziatives Array. Ein "normales" Feld, in dem also nur Daten übertragen werden, sieht wie folgt aus:

```
{"c":"7"}
```

Die Felderindikatoren wurden, um Traffic zu sparen, mit Kürzeln versehen. So steht beispielsweise c für content oder r für Rank. Mehr dazu bei den einzelnen Feldern. Als weitere Maßnahme könnten hier die 4 bis 5 Zeichen der Bool-Werte, die anzeigen ob ein Feldtyp aktiv ist, durch deren Zahlenrepräsentation ersetzt werden.

Die Formen sind in einer Datenbanktabelle abgespeichert, die über eine Benutzeroberfläche verwaltbar ist. In dieser Tabelle sind, neben dem Namen für die Verwaltung, folgende Informationen:

FIELD

Hier muss das Feld eingestellt werden, das aus der Ergebnistabelle ausgelesen wird. Bei gewissen Sonderfunktionen ist das nicht möglich oder nötig, wie z.B. beim Namen oder bei der Nation vom Teilnehmer.

Über eine Einstellung im Backend (geltend für die gesamte Form) kann die Zieltabelle für diese Einstellmöglichkeit ausgewiesen werden, da verschiedene Competitions verschiedene Ergebnistabellen haben. Der Eintrag, der in der betreffenden Tabelle in der betreffenden Zeile steht, wird in das "Content"-Feld vom Ergebnis geschrieben.

RANK

Der Rank hat eine besondere Bedeutung: Es gibt hier Sonderbehandlungen, damit nicht bei jedem neuen Rank die Ergebnisliste neu geladen werden muss.

Unter anderem wird hier auch eine Spezialbehandlung ausgeführt, falls der Status != "OK" ist. Das Feld wird dann nämlich mit dem Wert vom Status gefüllt (z.B. DNS ;)), aber trotzdem nach dem Rank sortiert, der in einem Meta-Wert abgespeichert wird.

Die Erstellung im Backend sieht aus wie folgt:

```
if ($m["JSON_RANK"] === "1")
{
  $thisfield[JSON_RANK] = true;
  $thisfield[JSON_CONTENT] = $row["Rank"];
  if($state_not_ok) $thisfield[JSON_STATE] = $state;
  if(isset($row["Tie"]) && $row["Tie"] > 0) $thisfield[JSON_TIE_RANK] = true;
}
```

Codebeispiel 29: Rank-Behandlung Serverseitig

Man sieht, wenn die Zeile in der Tabelle der Formen das Feld JSON_RANK mit dem Wert 1 belegt hat, wird diese Spezialbehandlung gestartet. Wenn der State des Teilnehmers nicht ok sein sollte, wird dieser in einem Extrafeld vermerkt, damit die Verarbeitung darauf eingehen kann. Wichtig ist noch der Vermerk ob der Teilnehmer sich seinen Platz mit einem anderen Teilnehmer teilt bzw. ob sich aufgrund von Punktegleichheit sein Platz verschieben könnte.

Das zugehörige JSON-Ergebnis sieht z.B. so aus:

```
{"c":"3", "r": true}
```

Somit kann das Frontend folgende Sonderbehandlung anwenden:

```
if(typeof cell[JSONDATA.RANK] == "boolean")
```

und die Daten weiterverarbeiten.
Die Verarbeitung der Daten wird genauer unter erklärt.

NAME

Wenn gesetzt, wird der Name aus der Competitor-Tabelle gelesen.
Dafür wird ein eigenes Objekt kreiert, das sich selbst aus der Datenbanktabelle erstellt und alle verfügbaren Informationen abspeichert. Das erfordert einen Datenbankaufruf pro Zeile der erstellten Ergebnisdatei. Aufgrund des *Cachings* ist dieser Punkt zu vernachlässigen - im Sinne von *"The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet."* [134].

Felder mit dem Typ `Name` haben clientseitig keine besondere Behandlung, da sie als purer Text behandelt werden. Dasselbe gilt für Felder wie `Round to one` oder `Nation`, die nur serverseitig eine Spezialbehandlung brauchen. Diese Felder werden auch im JSON nicht speziell markiert.

ROUND TO ONE

Wenn gesetzt, wird das Ergebnis aus der Tabelle nicht so genommen wie es ist, sondern auf eine Kommastelle abgerundet.

NATION

Diese Funktion liest die Nation aus der competitor-Tabelle. Dazu wird das gleiche Objekt verwendet wie für den Namen, und die Informationen werden für die nächste Verwendung gespeichert.

APPEND

Wenn dieses Feld gesetzt ist und ein weiterer Run von einem Competitor hereinkommt, wird dieses Feld nicht überschrieben (wie es z.B. beim Rank sinnvoll ist - denn der vorige Rank stimmt wahrscheinlich nicht mehr, der neue schon), sondern angehängt.

Vorher:

3	15

Ergebnis - Feld 1 ohne Append, Feld 2 mit:

2	15 18

Bei Append wird der folgende Code clientseitig ausgeführt:

```
            /**
              * text, where additional rows can be appended. used with
id and append for multiple runs
              */
            if(typeof cell[JSONDATA.APPEND] == "boolean")
            {
              row.append($("<td></td>").data('append',
true).html(content));
              return;

            }
```

Codebeispiel 30: Append

Mit dem Zusatz "append" als Boolean-Wert der jQuery Daten bekommt diese Zelle beim Hinzufügen eine gesonderte Behandlung.
Der Boolean-Wert `append` wird gesetzt, wenn in der Ziel-Tabelle eine Zeile gefunden

wird, deren ID dem aktuellen Competitor entspricht. Der Inhalt von jeder Zelle dieser gefundenen Zeile wird nun ersetzt - bis auf die Zellen, die mit dem Zusatz "append" versehen sind. Bei diesen Zellen wird der Inhalt der neuen Zeile mit einem Zeilenumbruch angefügt.

```
    /**
     * if we got 2 or more runs, we want to append that information to
the first run.
     */
    if(append)
    {
      if(value.find("#"+append_id).length > 0)
      {
        var children = value.find("#"+append_id).children();
        $(rowcopy).find("td").each(function(index, value){
          if($(children[index]).data('append'))
$(children[index]).append("<br>"+$(value).text());
          else $(children[index]).text($(value).text());
        });
```

Codebeispiel 31: Append im Einsatz

Der String-Wert `set_color` benötigt auch eine Spezialbehandlung. Wenn die Werte ersetzt oder angehängt werden, wird die vorher erstellte Zeile verworfen. Dadurch würde die gesetzte CSS-Klasse ebenso verloren gehen, weswegen sie hier explizit gesetzt werden muss.

```
        if(set_color != "")
value.find("#"+append_id).attr('class', set_color);
```

Der Boolsche Wert `is_partial_data` deutet darauf hin, ob die Daten komplett neu sind (wenn die Seite neu geladen wurde) oder ob zu existierenden Daten neue Daten hinzukommen. In diesem Fall sollte diese Zeile eine Markierung als "neu" bekommen.

```
        if(is_partial_data)
        {
            value.find("#"+append_id).effect("highlight",
{color: "red"}, 2500);
        }
```

Die Fernsehsprecher haben natürlich etwas anderes zu tun, als zum letzten Ergebnis zu scrollen. Daher gibt es eine Funktion, die den letzten Teilnehmer immer in Sichtweise hält, die später noch genauer beschrieben werden soll.

```
        display.guaranteeVisibility($("#"+append_id));
```

Das Tabellen-Sortier-Plugin benötigt eine Information, wenn sich etwas in der Tabelle getan hat. Das Sortieren wird dadurch noch nicht ausgelöst, das wird aus Performance-Gründen nach dem Fertigladen aller Ergebnisse erledigt.

```
        value.trigger("update");
      }
    }
```

Falls es sich nicht um eine existierende Zeile handelt, muss die erstellte Zeile in die Tabelle eingefügt werden. Dabei wird die Farbe gesetzt und das Sortier-Plugin über eine neue Zeile informiert.

```
    else if($(rowcopy).children().length > 1)
    {
      if(set_color != "") rowcopy.attr('class', set_color);
      value.append(rowcopy);
      value.trigger("update");
      value.trigger("appendCache");

  }
```

Codebeispiel 32: Append oder Replace beim Einfügen von Zeilen

Es folgen weitere Anweisungen, die in Spezialfällen Zeilen hervorheben, Tabellen als "aktualisiert" markieren und die Tabellen für eine leichtere Übersicht in Zebrastreifen - Farben unterteilen. Diese Zeilen bringen keine neuartige Funktionalität und werden als redundant ausgelassen.

MULTIPLE ROW

Manchmal sollen mehrere Ergebnisse eines Runs untereinander stehen ohne dass mehrere Runs gefahren werden. Das kann mit diesem Feld angezeigt werden.

Diese Funktion muss mit jedem Feld gesetzt werden, das untereinander angezeigt werden soll.

Üblicherweise werden Felder in eine Zelle pro Wert gelegt:

a	b	c	d

Wenn jetzt "b" und "c" als `Multiple Row` markiert werden, würde das Ergebnis so aussehen:

a	b c	d

MULTIPLE ROW END

Mit dem Ende muss markiert werden, welches der letzte Eintrag untereinander ist. `Multiple Row` muss dazu nicht gesetzt bleiben.Das ist notwendig, da sonst 2 `Multiple Row`s hintereinander nicht funktionieren würden.

So könnte aus

a	b	c	d

wenn "a" und "c" als `Multiple Row` definiert sind, und "b" und "d" als `Multiple Row End`, das Ergebnis so aussehen:

a b	c d

Der Code sieht aus wie folgt:

```
/**
 * multiple rows in one entry.
 */
if(typeof cell[JSONDATA.MULTIPLE_ROW] == "boolean")
{
   var html_entry = "";

   $.each(content, function(j, entry){
     html_entry += entry + "<br>";
   });
   row.append($("<td></td>").html(html_entry));
   return;
}
```

Codebeispiel 33: Einfügen von Multiple Rows

Die Einträge werden also als JSON-Array übergeben und einzeln in einen String zusammengefügt. Das Ergebnis wird dann in die Zelle eingetragen.

TIME ON COURSE

Dieses Feld ist nur gültig für das Oncourse-Fenster, das anzeigt, welche Teilnehmer gerade auf der Strecke sind. In dieses Feld schreibt das Zeitnehmungssystem die Anzahl an Millisekunden, die seit dem 01.01.1970 vergangen sind, für den gewöhnlichen Anwender also nicht besonders sinnvoll. Wenn dieses Feld aktiv ist, werden daraus automatisch die Sekunden berechnet und die Zeit, die der Teilnehmer auf der Strecke ist, wird jede Sekunde aktualisiert.

Für diesen Zweck wurde ein jQuery-Plugin geschrieben, das hier genauer erklärt wird. Zuerst wurde die Anpassbarkeit für mehrere Sprachen ermöglicht.

```
if(typeof lang === "undefined") var lang = {};
lang.time_prefixes = ['','','','','',''];
lang.time_suffixes = ['Y ','M ','d ',':',':',''];
```

In den folgenden Zeilen sieht man die Deklaration des Plugins und die Einbettung der Optionen. Über die jQuery-Funktion extend werden die Optionen mit einer höheren Priorität für die mitgegebenen Argumente mit den defaults vereint.

```
(function($) {

  $.fn.stopwatch = function(options)
  {

    var defaults =
    {
      prefix: lang.time_prefixes,
      suffix: lang.time_suffixes,
      limit: 2
    };

    var options = $.extend(defaults, options);
```

Das Kernstück des Plugins - die Schleife wird für jedes der betroffenen Elemente durchlaufen. Das Plugin wird über einen jQuery-Selektor aufgerufen, im Stil von "`$('div.class').stopwatch();`".

```
    return this.each(function()
    {
      /**
       * for future reference
       */
      var obj = $(this);

      /**
       * plugin settings
       */
      var limit = options.limit;
      var seconds_per_unit = [31536000, 2592000, 86400, 3600, 60, 1];
```

Im folgenden Code wird versucht, die jeweils größte Einheit von dem Wert abzuziehen, bis der Wert 0 erreicht. Der Code ist nur zum besseren Verständnis vom Gesamtbild abgebildet.

```
      /**
       * time.
       * expects seconds, gives back something like
prefix12suffixprefix05suffix
       */
      function time_format (seconds)
      {
        var index = 0, unit = 0, formatted = "";
        while(seconds > 0)
        {
          unit = Math.floor(seconds / seconds_per_unit[index]);
          if(unit > 0)
          {
            seconds -= seconds_per_unit[index] * unit;
            formatted += lang.time_prefixes[index] + unit +
lang.time_suffixes[index];
            if(--limit == 0) return formatted;
          }
          index ++;
        }

        return formatted;
      }
```

Codebeispiel 34: Hauptfunktion des Stopwatch-Plugins

Die folgende Funktion überwacht die Aktualisierung der Zeit jede Sekunde. Da ein Intervall von einer Sekunde äußerst selten genau zur Sekunde eintritt, und jedes Mal eine weitere Verzögerung dazukommt, wird die Startzeit eingespeichert und überprüft, ob diese Sekunde schon eine Änderung stattgefunden hat. Wenn ja, wird in einem kürzeren Zeitabstand die gleiche Abfrage nochmal durchgeführt.

```
      /**
       * change this thing now
       */
      function to_time ()
      {
        var str = obj.text();
        var seconds = str.replace(/\D/g,'');

        /**
         * shall we do something this second?
         */
        if(obj.data('actual_time') > 0)
        {

          if(new Date().getTime() == obj.data('actual_time'))
            {
              setTimeout(function(){
                obj.stopwatch();
              }, 100);
              return ;

              }
```

```
        }

        /**
         * we already modified this? well, better take the saved
value.
         */
        if(obj.data('seconds') > 0)
        {
          seconds = obj.data('seconds')*1 + 1;
        }

        var time_var = time_format(seconds);

        if(seconds > 3600) time_var = "";
       if (str != time_var)
        {
          obj.text(time_var);
          obj.data('seconds', seconds);
          obj.data('actual_time', new Date().getTime());
          setTimeout(function() {
            obj.stopwatch();
          }, 850);
        }
      }

      /**
       * only if it has content.
       */
      if ($(this).text().length > 0)
      {
        to_time();
        return this;
      }
    });
  };


})(jQuery);
```

Codebeispiel 35: Zeitinkrementierung

Der Code, der den Content einträgt und das Plugin startet, sieht aus wie folgt:

```
  /**
      * time on course - has to be counted up
      */
  if(typeof cell[JSONDATA.TIME_ON_COURSE] == "boolean")
  {
      var toc = $("<td></td>").html(Math.ceil((display.getTime() -
content*1000 - display.timedif)/1000));
      toc.stopwatch();
      row.append(toc);

      return;
  }
```

Codebeispiel 36: Zeitinkrementierung - Eintragung

Der Eintrag wird über die Multiplikation mit 1000 implizit zu einer Zahl konvertiert. Der Zeitunterschied (in Millisekunden gespeichert) wird von dem errechneten Unterschied abgezogen und das Ergebnis in das Plugin gefüttert.

TRACK COLOR

Wenn dieses Feld gesetzt ist, wird die Zeile mit der Klasse color+Inhalt des Feldes gefüllt, also z.b. class="colorR". Wenn dafür in der CSS Datei eine Farbe gesetzt ist, wird diese angezeigt. Das ist nötig für die Bracket-Runs, wo die Teilnehmer gleichzeitig auf der Strecke und nur nach der Farbe ihrer Trikots unterscheidbar sind.

Eine Besonderheit dabei ist, dass die Reihenfolge, in denen die Teilnehmer starten, immer definiert ist. Deswegen muss dem Sortier-Plugin ein Hinweis mitgegeben werden, welcher Track welche Reihenfolge darstellt. Die Eintragung der Zelle ist bis auf die zusätzlichen Daten gleich wie bei den anderen Zellen.

```
    /**
      * TRACK_COLOR of this competitor
      */
  if(typeof cell[JSONDATA.TRACK_COLOR] == "boolean")
  {
        set_color = "color"+content;

        /**
         * add content like normal text
         */
        var sort_order = 0;
        if(content == "R") sort_order = 1;
        if(content == "G") sort_order = 2;
        if(content == "B") sort_order = 3;
        if(content == "Y") sort_order = 4;

        row.append($("<td></td>").html(content).addClass("track")
.data('sort_order', sort_order));
        return;
  }
```

Codebeispiel 37: Track-Color

ALWAYS SHOW

Wenn der Status != "OK" ist, werden keine Felder bis auf Namen und Rank (der jetzt den State anzeigt, aber trotzdem nach Rank sortiert ist) und Nation angezeigt. Alle Felder, bei denen "Always Show" aktiv ist, werden auch angezeigt. Dabei handelt es sich um eine serverseitige Spezialbehandlung.

FORMAT TIME

Hier wird die Zeit serverseitig auf das Format M:SS.xx gebracht. Dabei musste ein Mix zwischen Berechnung und regular expressions verwendet werden, da sonst die ms fälschlicherweise auf- bzw. abgerundet wurden.

GER	1:09.72
CAN	1:09.88
SWE	1:09.91
CAN	1:09.95
USA	1:09.95
SLO	1:09.99
USA	1:10.02

Abbildung 30: Zeitformat

Als Beispiel, wie in der Schleife pro Reihe die Form durchgangen wird, wird hier das Zeitformat erstellt. Als erstes die Überprüfung, ob das Attribut Zeitformat für dieses Feld gesetzt ist. Dann die Abfrage des Datenbankfelds, das für dieses Feld gesetzt ist; gerundet bzw. moduliert für Minuten und Sekunden. Die Millisekunden werden nach einem gesetzten Punkt oder Komma (je nach Host) abgeschnitten.

```
if ($m["DB_FORMAT_TIME_M_S"] === "1")
 {
   $min = floor($row[$m["field"]] / 60);
   $sec = floor($row[$m["field"]] % 60);
   $ms = preg_replace('/.*\./','', $row[$m["field"]]);
   if($min < 10) $min = "$min";
   if($sec < 10) $sec = "0$sec";
   $thisfield[JSON_CONTENT] = "$min:$sec.$ms";
   if($state_not_ok) $thisfield[JSON_CONTENT] = "";
 }
```

Codebeispiel 38: Zeitformatierung

SICHTBARKEITSGARANTIE

Wie bei "Append" vermerkt, gibt es eine kleine Funktion, die dafür sorgt, dass der zuletzt eingetragene Teilnehmer automatisch in das Sichtfeld gescrollt wird. Wenn zu viele Teilnehmer bei einem Rennen sind, kann im CIS das Anzeigefenster nicht mehr alle Teilnehmer anzeigen[12]. Ist nun der letzte Teilnehmer nicht mehr sichtbar, hätte der Sprecher keine Informationen.

Daher wird bei jedem neuen Teilnehmer eine Funktion aufgerufen, die für dieses bestimmte Fenster einträgt, welcher der letzte Teilnehmer war.

Die meiste Arbeit, die diese Funktion verrichtet, bezieht sich auf unitialisierte Variablen, die wahre Funktion wird mit einem `Push` erledigt.

```
display.guaranteeVisibility = function(jrow)
{
  if(display.is_livescoring) return;
  if(display.actual_window === undefined) return;
  if(display.type != display.WINDOWTYPE_RESULT) return;
  if(display.actual_window["alwaysvisible"] === undefined)
display.actual_window["alwaysvisible"] = [];

  display.actual_window["alwaysvisible"].push(jrow);
};
```

Codebeispiel 39: Sichtbarkeitsgarantie - Eintragung

Nachdem das Laden abgeschlossen ist, wird die Funktion aufgerufen, die die Sichtbarkeit umsetzt.

Falls Scrollen abgeschaltet ist (Zeile 2 der Funktion), braucht die Funktion nichts zu tun - ebenso, wenn keine Daten vorhanden sind.

Die Zeilen werden über eine CSS-Klasse verkleinert, nicht komplett entfernt. Eine Entfernung aus der Tabelle würde das Sortier-Plugin durcheinanderbringen und die Türen für ungewollte Side-Effects öffnen.
Die erste versteckte Zeile bekommt eine besondere Markierung mit der Klasse "first", damit sie visuell anzeigt, dass hier versteckte Zeilen sind.

[12] Das Livescoring soll immer alle Informationen anzeigen.

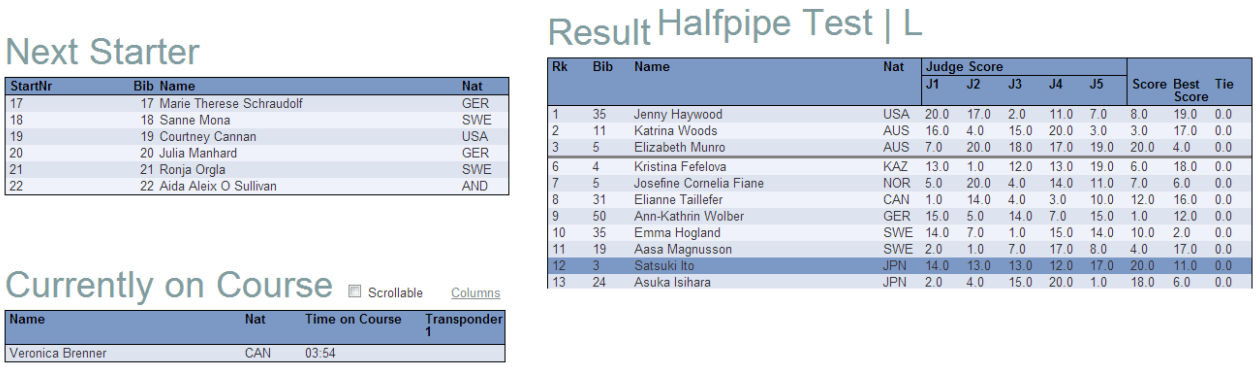


Abbildung 31: Beispiel für versteckte Zeilen und einem markierten letzten Teilnehmer

Diese Markierungen sowie die Markierung für den letzten Teilnehmer werden entfernt. Daraufhin wird dem letzten Teilnehmer die Markierung "lastCompetitor" wieder hinzugefügt, sowie in einer Schleife solange Teilnehmer entfernt (maximal 50 mal, um eine Endlosschleife zu verhindern), bis dieser Teilnehmer sichtbar ist.

```
display.guaranteeVisibilityScroller = function()
{
  if(display.actual_window === undefined) return;
  if(display.getTable().parents(".cis-window").find(".toggleScrollMe
input").is(":checked") == false) return;
  if(display.actual_window["alwaysvisible"] === undefined) return;

display.getTable().find("tr").removeClass("hiddenRow").removeClass("fi
rst").removeClass("lastCompetitor");
  value =
$(display.actual_window["alwaysvisible"][display.actual_window["always
visible"].length - 2]);
  if(value.length <= 0) return;
  value.addClass("lastCompetitor");

  var i = 50;
  while(value.position()["top"] + value.height()*2 >
value.parents(".cis-window").height())
  {
    display.removeOneLineForVisibility();
    if(i-- <= 0) break;
  }
  display.getTable().find("tr.hiddenRow:first").addClass("first");
};
```

Codebeispiel 40: Sichtbarkeitsgarantie für die Ergebnistabelle

In der Funktion, die Zeilen entfernt, müssen noch Spezialfälle abgedeckt werden. Es sollen immer die ersten 3 Ergebnisse sichtbar bleiben und bereits versteckte Zeilen sollen nicht noch einmal versteckt werden.

```
display.removeOneLineForVisibility = function()
{
  table = display.getTable();
  $.each(table.find("tbody tr"), function(index, line){
    if($(line).find(".rank").data("rank") <= 3) return true;
    if($(line).hasClass("hiddenRow")) return true;
    $(line).addClass("hiddenRow");
    return false;
  });

};
```

Codebeispiel 41: Sichtbarkeitsgarantie - Zeilenentfernung

AUSGABE

Auf der Clientseite wird dann für jede Ergebniszeile[13] eine Schleife durch alle Ergebnisse gemacht. Pro Feld wird unterschieden, ob es eine Spezialbehandlung benötigt oder nicht - und während dieser Schleife wird eine Tabellenzeile zusammengebaut.

Den besten Einblick bringt ein Beispiel, deswegen hier der Code zur Erstellung einer Rank-Zeile - die aufwendigste Spezialbehandlung.

Die folgenden Punkte haben sich als wichtig herausgestellt:

- Wenn ein Teilnehmer schon in dieser Tabelle steht, bedeutet das, dass in diesem Heat zwei Runs stattfinden. Diese Runs sollten also untereinander angezeigt werden.
- Wenn ein Rank bereits vorhanden ist (und es sich nicht um einen TIE handelt), müssen die Ranks, die gleich oder höher sind, inkrementiert werden.
- Wenn ein Teilnehmer schon in dieser Tabelle steht, könnte dieser den gleichen Rank haben. Daher muss dieser Rank von der Suche nach einem gleichen oder

[13] Pro Run: Ein Assoziatives Array

höheren Rang ausgeschlossen werden (der Text durch einen leeren String ersetzt werden).

- Die Zelle benötigt eine spezielle Markierung, damit sie später gefunden werden kann. Das passiert hier über die Klasse "rank", da JQuery hier eine leichte spätere Auffindbarkeit über Selectoren [135] ermöglicht.
- Der Rank darf nicht nur als Text abgespeichert werden. Die Funktion **`.data`** von jQuery [136] gibt uns hier die Flexibilität, in der Zelle weitere Informationen abzuspeichern ohne das DOM mit unnützen Informationen zu belasten.
- Wenn ein Rank als `TIE_RANK` markiert ist und die Daten gerade inkrementell geladen wurden, müssen wir von fehlerhaften Daten ausgehen. Aufgrund der Vielfältigkeit der Regeln kann hier keine Behandlung auf Anzeigeseite stattfinden. Daher werden alle Daten dieses Elements neu vom Server geladen.
- Wenn ein Teilnehmer einen Status hat, der nicht "OK" ist, wird dieser vom Backend mitgeschickt. Das bedeutet, der Teilnehmer hat sich disqualifiziert, ist nicht gestartet oder hat das Rennen nicht beendet. In diesem Fall sollen nur noch sehr wenige Daten angegeben werden; das sind meistens der Rang, die Startnummer, der Name und das Herkunftsland. Siehe *Always Show*.

```
/**
 * rank text. like normal text,
 * but if the rank is doubled, we got to recalculate that.
 */
if(typeof cell[JSONDATA.RANK] == "boolean")
{
	rank = parseInt(content);
	var rank_to_change = rank;


	/**
	 * replace with nothing, count up all ranks until previous place
("") is reached
	 */
	if(append)
	{
		if(table.find("#"+append_id + " .rank").length > 0)
		{
			table.find("#"+append_id + " .rank").text("");
			table.find("#"+append_id + " .rank").data('rank',
rank);
		}
	}

	/**
	 * note: 2 competitors with same rank can exist
	 */
	if(typeof cell[JSONDATA.TIE_RANK] == "undefined" && rank != 0)
```

```
	{
		var do_later = [];
		table.find("td.rank").each(function()
		{
			var rank_plus = 0;
			table.find("td.rank").each(function()
			{
				//if this rank does not exist, it means we just
changed place with someone
				if(parseInt($(this).text()) == rank_to_change)
				{
					do_later.push($(this));
					rank_plus++;
				}
			});
			rank_to_change +=rank_plus;
		});

		$.each(do_later, function(index, value){
			if(value.data('state') != true)
			{
				value.text(parseInt(value.text())+1);
				value.data('rank', parseInt(value.data('rank')
+ 1));
			}
		});
	}

	if(typeof cell[JSONDATA.TIE_RANK] == "boolean")
reload_if_partial_data = true;

	var td = $("<td class=\"rank\"></td>").data('rank', rank);
	if(rank > 0) td.html(rank);

	if(typeof cell[JSONDATA.STATE] == "string")
	{
		td.html(cell[JSONDATA.STATE]);
		td.data('state', true);
	}
	row.append(td);
	return;
}
```

Codebeispiel 42: Rank-Behandlung Clientseitig

Diese fertig generierte Reihe wird dann in die definierte Tabelle eingeschrieben. Die Sortierung in der Tabelle wird danach über ein jQuery-Plugin durchgeführt, dass eine benutzerdefinierte Sortierung durch einen Klick auf den Tabellennamen erlaubt.

BREITE

Um die Spaltenbreite leicht anpassbar zu machen, wurde auf eine Hybridlösung zurückgegriffen. Die Speicherung in der Datenbank erfolgt in absoluten Pixeln, ausgehend von der aktuellen Bildschirmbreite. Damit kann beim Ändern der Breite immer die exakt gleiche Ansicht gezeigt werden, ohne hier Rundungsfehler oder Ähnliches einberechnen zu müssen, und man kann die Übersicht am leichtesten behalten.

Abbildung 32: Breitenverwaltung

Bei der Ausgabe haben Pixelangaben natürlich weniger Sinn, da ein CIS-Fenster z.B. oft kleiner sein wird als die Breite des Fensters. Daher wird bei der Ausgabe der Gesamtwert der Pixel addiert und die Breite dann weiterfolgend in % ausgegeben.

BRACKETS

Eine ganz besondere Form der Ausgabe sind die Brackets. Diese Anzeigeform ist für Skicross und Dual Moguls wichtig, da der Sprecher hier ganz andere Informationen benötigen würde. Zur übersichtlichen Darstellung soll jeder Run in einer eigenen Tabelle gezeigt werden, mit einer Art Fortschrittsansicht.

Das bedeutet, Läufer die in einem Semifinale weiterkommen (Knock-Out-Prinzip), stehen durch das Erreichen von Platz 1-2 im Finale. Diese Anzeige würde den bisher implementierten Prinzipien widersprechen. Eine Lösung dafür war, eine spezielle Anzeigeform in einem Tab bzw. in einem Window zu finden, die die Zuordnung zu den Tabellen selbst verwaltet und dennoch vom selben System gefüttert wird. So wurden die Filter aus den Ergebnistabellen entfernt und alle Daten an das Javascript geschickt.

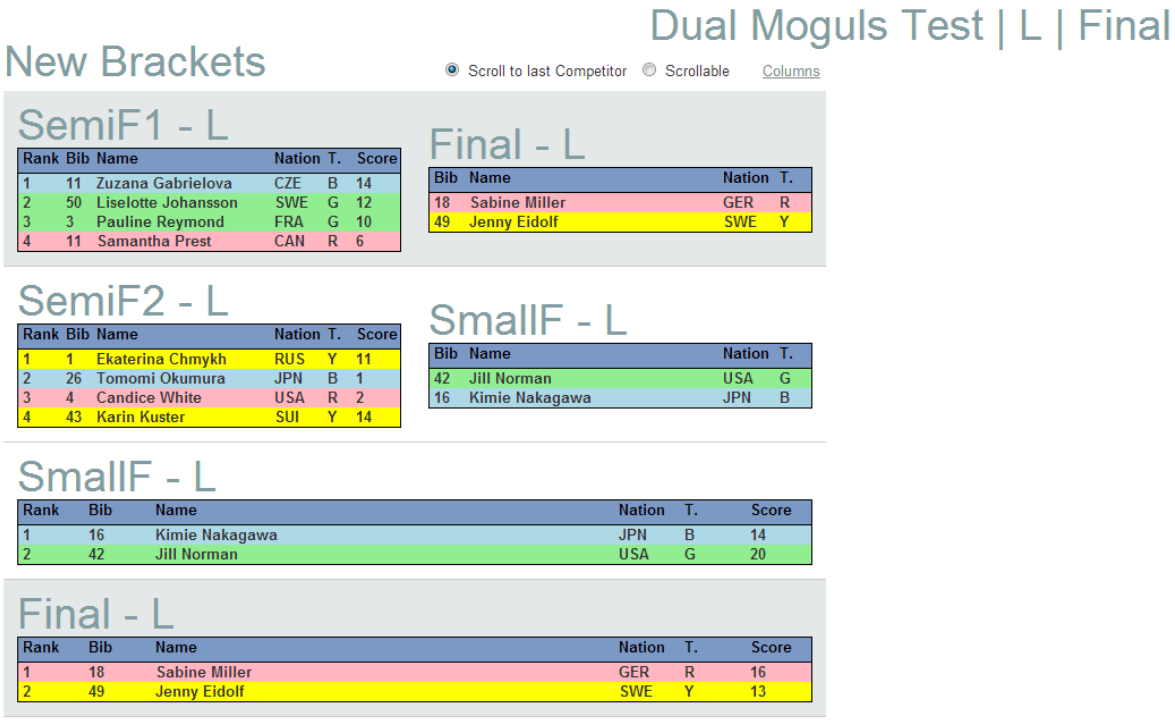
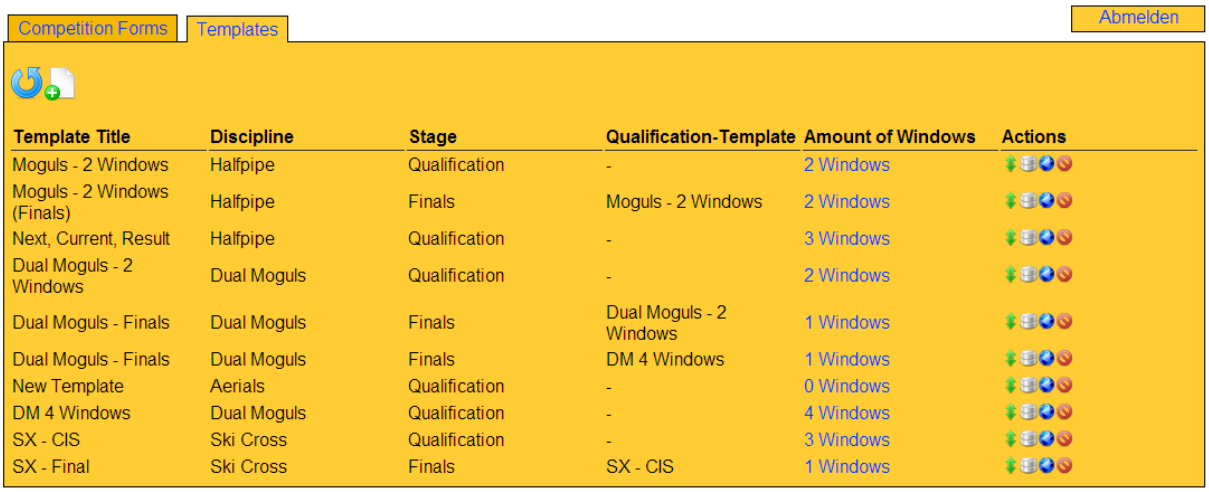


Abbildung 33: Brackets

Diese Brackets ersetzen alle sonstigen Anzeigen im CIS, da diese nicht benötigt werden. Hier kommt ins Spiel, dass Templates für Qualifikation und Finale extra eingestellt werden können. Brackets sind immer eine Form, die im Finale vorkommt. Gefüllt werden die Brackets über das gleiche System wie die anderen Ergebnistabellen. Ein großer Unterschied ist allerdings, das Teilnehmer, die weiter sind, in zwei Tabellen eingetragen werden müssen - in die Fortschrittstabelle (Siehe Abbildung 12: Final - L, oben rechts) und in die Ergebnistabelle (Abb. 12, Final - L, unten). Durch eine Schleife über alle betroffenen Tabellen benötigt das keine weitere Spezialbehandlung. Der Unterschied, dass in der Fortschrittstabelle kein Rang und keine Punkte angezeigt werden, wird über das Column-Hiding erreicht. Für alle rechts stehenden Tabellen in einem Bracket-Container wird die erste und die letzte Spalte versteckt - und damit das Ziel einer Fortschrittstabelle in kurzer Zeit erreicht.

TEMPLATES

Das Herzstück vom CIS sind die Templates. Jedes Template kann mehrere Fenster beinhalten. Diese werden in einer Verwaltung generiert und über eine WYSIWYG-Ansicht in der Größe und Position angepasst.



Template Title	Discipline	Stage	Qualification-Template	Amount of Windows	Actions
Moguls - 2 Windows	Halfpipe	Qualification	-	2 Windows	
Moguls - 2 Windows (Finals)	Halfpipe	Finals	Moguls - 2 Windows	2 Windows	
Next, Current, Result	Halfpipe	Qualification	-	3 Windows	
Dual Moguls - 2 Windows	Dual Moguls	Qualification	-	2 Windows	
Dual Moguls - Finals	Dual Moguls	Finals	Dual Moguls - 2 Windows	1 Windows	
Dual Moguls - Finals	Dual Moguls	Finals	DM 4 Windows	1 Windows	
New Template	Aerials	Qualification	-	0 Windows	
DM 4 Windows	Dual Moguls	Qualification	-	4 Windows	
SX - CIS	Ski Cross	Qualification	-	3 Windows	
SX - Final	Ski Cross	Finals	SX - CIS	1 Windows	

Diese Templates werden einer Disziplin zugeordnet und einer Stage bzw. einem Qualifikations-Template. Dadurch kann, wenn für die Finalrennen eine andere Ansicht gewünscht ist als für die Qualifikation, das Template gewechselt werden. Dafür sorgt ein System, das über periodische Abfragen prüft, ob der momentan aktuellste Run angezeigt wird.

Das zu verwendende Template kann in einer Liste ausgewählt werden bzw. über einen Link mitgegeben werden. Damit könnte der Veranstalter über einen Link auch festlegen, wer welche Informationen bekommt.

In einer weiterführenden Überlegung könnte man für einzelne Templates eine Zeitverzögerung einbauen, was gerade bei Rennen, die von Wetten erfasst sind, wichtig sein könnte.

Über die zweite Verwaltungsansicht kann eingestellt werden, welches Fenster wo und in welcher Größe sichtbar sein wird.

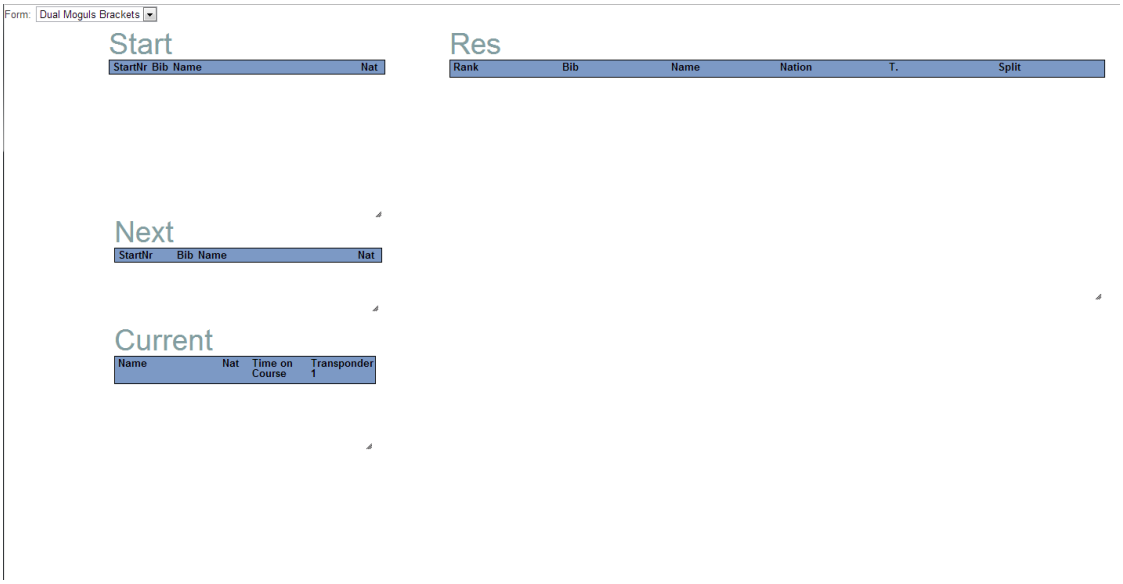


Abbildung 34: Template-Verwaltung: Fenstergröße

WINDOWS

Die Windows stellen je eine logische Einheit dar. Diese Einheit wird vom Template in ihrer Größe und Position definiert; ebenso bekommt das Window einen Typ zugewiesen. Wahlweise kann das Window die eingestellte Form (Qualifikation und Final) von der Competition überschreiben.

Das Window teilt bei der Generierung dem Javascript mit, dass es anwesend ist; es gibt dem Javascript die Form, den Typ, die ID vom Container und der Competition mit. Diese Daten werden als Objekt in ein Array gespeichert.
Das Livescoring-Javascript macht nun eine Abfrage an den Server. Bei allen Abfragen wird als erstes das aktuelle Objekt hinten an das Array angehängt und der vorderste Eintrag ausgelesen. Dadurch wird reihum jedes Fenster aktualisiert, und die Fenster stellen beim Wechsel selbst den Request um, damit die Informationen ankommen, die das Fenster braucht.

Die Implementation der Windows selbst ist nicht weiter beachtenswert, es handelt sich um absolut positionierte DIV's mit wahlweise aktiviertem Scrolling oder Abschneiden der überstehenden Elemente.

ERGEBNISSE

Die Ergebnisse der Arbeit sind unter http://masterarbeit.midlight.eu/ einsehbar. Auf der Seite wird ein Link zu einer Kontrollseite und zu Anzeigenseiten angezeigt, mit der eine vorgefertigte Veranstaltung manipuliert werden kann.

ZUSAMMENFASSUNG UND AUSBLICK

Durch diese Arbeit hat sich nicht nur mein Wissen über Skirennen stark verbessert. Die verwendeten Techniken waren mir Großteils schon bekannt, aber in so einer Art und Weise habe ich sie noch nie zusammengesetzt.
Ich konnte viel Erfahrung gewinnen was die mehrstufige Anpassung von Anzeigen angeht. Auch wenn die Sicherheit bei diesem Thema vollends ausgeklammert wurde und dadurch noch eine Ebene an Komplexität ignoriert wurde, ist auf dieser Basis leicht etwas Zusätzliches aufzubauen.
Mit diesem fast generischen Aufbau lassen sich viele dynamische Daten problemlos anzeigen, auch wenn eine Spezialisierung in Richtung von Punkten und Sortierung forciert wurde. So könnten sich auch Punkte von Spielen oder ähnlichem in diesen Anzeigecontainern anzeigen lassen.
Die mir bekannten Zeitnehmer waren von diesem Projekt begeistert.

LITERATURVERZEICHNIS

[1] "Fis-Ski - Internationaler Ski Verband," [Online]. Available: http://www.fis-ski.com/. [Accessed Dezember 2012].

[2] "Google Chrome," Google, [Online]. Available: http://www.google.com/intl/de/chrome/browser/. [Accessed Dezember 2012].

[3] "Mozilla Firefox," Mozilla, [Online]. Available: https://www.mozilla.org/de/firefox/. [Accessed Dezember 2012].

[4] "Opera Browser," Opera, [Online]. Available: http://de.opera.com/. [Accessed Dezember 2012].

[5] "Internet Explorer 9," Microsoft, [Online]. Available: http://www.microsoft.com/de-at/windows/internet-explorer/. [Accessed Dezember 2012].

[6] "StatCounter," StatCounter, 10 10 2010. [Online]. Available: http://gs.statcounter.com/#mobile_vs_desktop-ww-monthly-201010-201010-bar. [Accessed 02 2013].

[7] "StatCounter," StatCounter, 31 12 2012. [Online]. Available: http://gs.statcounter.com/#mobile_vs_desktop-ww-monthly-201212-201212-bar. [Accessed 02 2012].

[8] "Go-Globe," Go-Globe.com, 06 09 2012. [Online]. Available: http://www.go-globe.com/blog/mobile-web-traffic/. [Accessed 02 2013].

[9] "Statistik Austria - die Informationsmanager," Statistik Austria, 01 2013. [Online]. Available: http://www.statistik.gv.at/web_de/statistiken/informationsgesellschaft/index.html. [Accessed 02 2013].

[10] "The WebKit Open Source Project," WebKit, 06 2012. [Online]. Available: http://www.webkit.org/. [Accessed 02 2013].

[11] "Google Chrome Developers Page," Google, 23 01 2013. [Online]. Available: https://developers.google.com/chrome-developer-tools/docs/overview. [Accessed 02 2013].

[12] "Safari Dev Center," Apple Inc., 2013. [Online]. Available: https://developer.apple.com/devcenter/safari/index.action. [Accessed 02 2013].

[13] "Google Groups," Google, 2012. [Online]. Available: http://productforums.google.com/forum/#!topic/analytics/YVevuYq1Lo4. [Accessed 02 2013].

[14] M. Firtman, "Preinstalled Browsers," in *Programming the Mobile Web, 2nd Edition*, USA, O'Reilly Media, Inc., 2013, pp. 51-55.

[15] M. Firtman, "Rendering Engines," in *Programming the Mobile Web, 2nd Edition*, USA, O'Reilly Media Inc., 2013, pp. 35-40.

[16] WebAIM, "WebAIM: Screen Reader User Survey," WebAIM, 2012. [Online]. Available: http://webaim.org/projects/screenreadersurvey4/#javascript. [Accessed 02 2013].

[17] Xeoncross, "How powerful are mobile browsers compared to desktops?," 05 2012. [Online]. Available: http://stackoverflow.com/questions/10489901/how-powerful-are-mobile-browsers-compared-to-desktops/10489942#10489942. [Accessed 02 2013].

[18] W. Leonhard, "Surprising winner in JavaScript speed wars: Metro IE10," InfoWorld, 06 2012. [Online]. Available: http://www.infoworld.com/t/microsoft-windows/surprising-winner-in-javascript-speed-wars-metro-ie10-196832. [Accessed 02 2013].

[19] e. International, "ECMA-262," 06 2011. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf. [Accessed 02 2013].

[20] "XMLHttpRequest: Test Coverage," W3C, 2012. [Online]. Available: https://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html. [Accessed 02 2013].

[21] S. McConnel, "Rapid Development," in *Rapid Development - Taming wild Software Schedules*, Canada, Microsoft Press, 1996, pp. 335-341.

[22] G. Schlossnagle, "Advanced PHP Programming," in *Advanced PHP Programming - A practical guide to developing large-scale Web sites and applications with PHP 5*, United States of America, Sams Publishing, 2004, pp. 244-250.

[23] The Apache Software Foundation, "Redirecting and Remapping with mod_rewrite," [Online]. Available: http://httpd.apache.org/docs/2.2/rewrite/remapping.html#fallback-resource. [Accessed November 2012].

[24] B. McGehee, "SQL Server Performance," [Online]. Available:

http://www.sql-server-performance.com/2007/views-general/. [Accessed November 2012].

[25] Oracle, "MySQL 5.1 Referenzhandbuch," [Online]. Available: http://dev.mysql.com/doc/refman/5.1/en/stored-routines.html. [Accessed November 2012].

[26] J. D. Zawodny and D. J. Balling, "High Performance MySQL," in *High Performance MySQL - Optimization, Backups, Replication & Load Balancing*, Canada, O'Reilly, 2004, pp. 80-85.

[27] Google, "Google Developers: Make the Web faster - Optimize caching," 03 2012. [Online]. Available: https://developers.google.com/speed/docs/best-practices/caching. [Accessed 02 2013].

[28] W3c, "HTTP - Hypertext Transfer Protocol," 03 2013. [Online]. Available: http://www.w3.org/Protocols/. [Accessed 02 2013].

[29] K. McArthur, "Pro PHP," in *Pro PHP - Patterns, Frameworks, Testing and more*, United States of America, Apress, 2008, pp. 273-284.

[30] J. Tidwell, "COMMON GROUND: A Pattern Language for Human-Computer Interface Design," 1998. [Online]. Available: http://www.mit.edu/~jtidwell/common_ground.html. [Accessed 02 2013].

[31] J. Tidwell, "Stack of Working Surfaces," 1999, 05. [Online]. Available: http://www.mit.edu/~jtidwell/language/stack_of_working_surfaces.html. [Accessed 02 2013].

[32] J. Tidwell, "Pile of Working Surfaces," 05 1999. [Online]. Available: http://www.mit.edu/~jtidwell/language/pile_of_working_surfaces.html.

[Accessed 02 2013].

[33] A. Cooper, R. Reimann and D. Cronin, "About Face 3," in *About Face 3 - The Essentials of Interaction Design*, United States of America, Wiley Publishing, Inc., 2007, pp. 178ff; 243-247; 283-285.

[34] K. Perzel and D. Kane, "Usability Patterns for Applications on the World Wide Web," 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.4303&rep=rep1&type=pdf. [Accessed 02 2013].

[35] M. v. Welie and H. Troetteberg, "INTERACTION PATTERNS IN USER INTERFACES," 2000. [Online]. Available: http://www.welie.com/papers/PLoP2k-Welie.pdf. [Accessed 02 2013].

[36] M. v. Welie, "Welie.com: Patterns in Interaction Design - Grid-based Layout," 2007. [Online]. Available: http://www.welie.com/patterns/showPattern.php?patternID=grid-based-layout. [Accessed 02 2013].

[37] W. Stewart, "The World Wide Web (WWW)," 2000. [Online]. Available: http://www.livinginternet.com/w/w.htm. [Accessed 02 2013].

[38] W3c, "W3C Recommendations Reduce 'World Wide Wait'," 08 07 1999. [Online]. Available: http://www.w3.org/Protocols/NL-PerfNote.html. [Accessed 02 2013].

[39] "MIT Computer Science and Artificial Intelligence Laboratory," 02 2013. [Online]. Available: http://www.csail.mit.edu/. [Accessed 02 2013].

[40] Inria, "Inria: Inventeurs du monde numérique," 02 2013. [Online]. Available:

http://www.inria.fr/. [Accessed 02 2013].

[41] Keio University, "Keio University," 02 2013. [Online]. Available: http://www.keio.ac.jp/. [Accessed 02 2013].

[42] L. Montulli and D. Kristol, "HTTP State Management Mechanism," 02 1997. [Online]. Available: http://www.w3.org/Protocols/rfc2109/rfc2109. [Accessed 02 2013].

[43] Google, "V8 JavaScript Engine," 02 2013. [Online]. Available: http://code.google.com/p/v8/. [Accessed 02 2013].

[44] D. Doge, "Google Instant technology advances, not just UI changes," 09 2010. [Online]. Available: http://dondodge.typepad.com/the_next_big_thing/2010/09/google-instant-technology-advances-not-just-ui-changes.html. [Accessed 02 2013].

[45] Google, "Google," 02 2013. [Online]. Available: https://www.google.at/. [Accessed 02 2013].

[46] W3c, "THE XMLHTTPREQUEST OBJECT PUBLICATION HISTORY," 02 2013. [Online]. Available: http://www.w3.org/standards/history/XMLHttpRequest. [Accessed 02 2013].

[47] B. Gibson, "JavaScript and AJAX Accessibility," 2006. [Online]. Available: http://www.w3.org/2006/Talks/0524-www-AjaxWAI.pdf. [Accessed 02 2013].

[48] "wcschools.com: XSLT Browsers," 01 2013. [Online]. Available: http://www.w3schools.com/xsl/xsl_browsers.asp. [Accessed 02 2013].

[49] "XSL Transformations (XSLT)," 11 1999. [Online]. Available: http://www.w3.org/TR/1999/REC-xslt-19991116. [Accessed 2 2013].

[50] "XSLT Tryit Editor," 01 2013. [Online]. Available: http://www.w3schools.com/xsl/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog. [Accessed 02 2013].

[51] "Grundlagen von XSL/XSLT," SelfHTML, [Online]. Available: http://de.selfhtml.org/xml/darstellung/xslgrundlagen.htm. [Accessed 02 2013].

[52] F. Siepmann, "Message Oriented Middleware am Beispiel von XMLBlaster," 06 2005. [Online]. Available: http://www.techfak.uni-bielefeld.de/~swrede/xml-isy/talks/mom-xmlblaster.pdf. [Accessed 02 2013].

[53] G. Napierala and M. Kubasch, "Programming-Wiki," 01 2012. [Online]. Available: http://programmingwiki.de/Einf%C3%BChrung_in_die_XML-Technologien_XPath,_XSLT_und_XQuery/XSLT. [Accessed 02 2013].

[54] J. Winkler, "XSLT: Programmieren mit XSLT," HTMLWorld, [Online]. Available: http://www.html-world.at/program/xslt_3.php. [Accessed 02 2013].

[55] "Using XML, XSLT and c# to create an RTF or PDF," 11 2009. [Online]. Available: http://stackoverflow.com/questions/1674257/using-xml-xslt-and-c-sharp-to-create-an-rtf-or-pdf. [Accessed 02 2013].

[56] SELFHTML, "Beispiele für XSLT," SELFHTML, 2006. [Online]. Available: http://de.selfhtml.org/xml/darstellung/xsltbeispiele.htm. [Accessed 02 2013].

[57] J. Allen, "Unofficial MSXML XSLT FAQ," 30 1 2001. [Online]. Available:

http://web.archive.org/web/20060428015421/http://www.netcrucible.com/xslt/msxml-faq.htm. [Accessed 02 2013].

[58] "The Apache Cocoon Project," Apache, 01 2005. [Online]. Available: http://cocoon.apache.org/2.1/index.html. [Accessed 02 2013].

[59] "Symphony: XSLT-powered open source content management system," [Online]. Available: http://getsymphony.com/. [Accessed 02 2013].

[60] D. Novatchev, "Stackoverflow: Is using XML and XSLT a good way to make a webpage?," 08 2012. [Online]. Available: http://stackoverflow.com/questions/3467410/is-using-xml-and-xslt-a-good-way-to-make-a-webpage. [Accessed 02 2013].

[61] G. Borkowski, "The Death of XSLT in Web Frameworks," 21 04 2009. [Online]. Available: http://java.dzone.com/news/death-xslt-web-frameworks. [Accessed 02 2013].

[62] "Is there a point creating a site using XSLT," 2009. [Online]. Available: http://stackoverflow.com/questions/560635/is-there-a-point-creating-a-site-using-xslt. [Accessed 02 2013].

[63] "What is preventing widespread use of XSLT for webpages?," 09 2012. [Online]. Available: http://stackoverflow.com/questions/1268329/what-is-preventing-widespread-use-of-xslt-for-webpages. [Accessed 02 2013].

[64] "Why has XSLT never seen the popularity of many other languages that came out during the internet boom?," 04 2012. [Online]. Available: http://stackoverflow.com/questions/77342/why-has-xslt-never-seen-the-popularity-of-many-other-languages-that-came-out-dur. [Accessed 02 2013].

[65] B. Bos, "CSS & XSL," W3C, 07 1999. [Online]. Available: http://www.w3.org/Style/CSS-vs-XSL. [Accessed 02 2013].

[66] P. Hégaret, "Document Object Model (DOM)," W3C, 06 01 2009. [Online]. Available: http://www.w3.org/DOM/. [Accessed 02 2013].

[67] M. Firtman, "Mobile HTML5," 06 02 2013. [Online]. Available: http://mobilehtml5.org/. [Accessed 02 2013].

[68] D. Crockford, "JSON: The Fat-Free Alternative to XML," 2006. [Online]. Available: http://www.json.org/fatfree.html. [Accessed 02 2013].

[69] "php.net: SimpleXML," 02 2013. [Online]. Available: http://php.net/manual/de/book.simplexml.php. [Accessed 02 2013].

[70] "php.net: The DOMElement class," 02 2013. [Online]. Available: http://php.net/manual/de/class.domelement.php. [Accessed 02 2013].

[71] "php.net: json_encode," 02 2013. [Online]. Available: http://php.net/manual/de/function.json-encode.php. [Accessed 02 2013].

[72] "XML Parser," w3schools, [Online]. Available: http://www.w3schools.com/Xml/xml_parser.asp. [Accessed 02 2013].

[73] "JSON Parser," wc3schools.com, [Online]. Available: http://www.w3schools.com/json/json_eval.asp. [Accessed 02 2013].

[74] "JSON Tutorial," w3schools.com, [Online]. Available: http://www.w3schools.com/json/default.asp. [Accessed 02 2013].

[75] Luke101, "Why is Everyone Choosing JSON Over XML for jQuery?," 2009. [Online]. Available: http://stackoverflow.com/questions/1743532/why-is-everyone-choosing-json-over-xml-for-jquery. [Accessed 02 2013].

[76] geme_hendrix, "Is parsing JSON faster than parsing XML," 2010. [Online]. Available: http://stackoverflow.com/questions/4596465/is-parsing-json-faster-than-parsing-xml. [Accessed 02 2013].

[77] D. Megginson, "All markup ends up looking like XML," 01 2007. [Online]. Available: http://quoderat.megginson.com/2007/01/03/all-markup-ends-up-looking-like-xml/. [Accessed 02 2013].

[78] "Think2Loud: JSON vs. XML: What Should You Use?," 12 2012. [Online]. Available: http://think2loud.com/680-json-xml/. [Accessed 02 2013].

[79] T. Bray, "JSON and XML," 12 2006. [Online]. Available: http://www.tbray.org/ongoing/When/200x/2006/12/21/JSON. [Accessed 02 2013].

[80] L. Borek, "Poor performance of XML parsing," 02 2013. [Online]. Available: http://forum.jquery.com/topic/poor-performance-of-xml-parsing. [Accessed 02 2013].

[81] P. P. Kock, ppk on JavaScript, New Riders, 2006.

[82] P.-P. Koch, "quirksmode.org: Kompatibilitätstabellen," 02 2013. [Online]. Available: http://www.quirksmode.org/compatibility.html. [Accessed 02 2013].

[83] P.-P. Koch, "The AJAX response: XML, HTML, or JSON?," 2005. [Online]. Available:

http://www.quirksmode.org/blog/archives/2005/12/the_ajax_respon.html. [Accessed 02 2013].

[84] A. Deveria, "Can I use... JSON parsing," 01 2013. [Online]. Available: http://caniuse.com/#cats=JS_API&agents=All. [Accessed 02 2013].

[85] Microsoft, "msdn: DataGrid Class," Microsoft, 07 2011. [Online]. Available: http://msdn.microsoft.com/en-us/library/system.windows.controls.datagrid(v=vs.95).aspx. [Accessed 02 2013].

[86] R. E. Johnson, "COMMUNICATIONS OF THE ACM: Frameworks= (Components+Patterns) - How frameworks compare to other," 19 1997. [Online]. Available: http://www.inf.ufsc.br/~vilain/framework-thiago/p39-johnson.pdf. [Accessed 02 2013].

[87] J. Vepsäläinen, "jster: We are an online catalog with 1025 javascript libraries and tools for web development inside," 02 2013. [Online]. Available: http://jster.net/. [Accessed 02 2013].

[88] A. Spetik, "Prototype VS jQuery - Strengths and Weaknesses?," 04 2010. [Online]. Available: http://stackoverflow.com/questions/2644556/prototype-vs-jquery-strengths-and-weaknesses. [Accessed 02 2013].

[89] jsjoe, "Is there any reason to prefer Prototype to JQuery?," 09 2009. [Online]. Available: http://stackoverflow.com/questions/1426023/is-there-any-reason-to-prefer-prototype-to-jquery. [Accessed 02 2013].

[90] P. Mortensen, "Why does everyone like jQuery more than prototype/script.aculo.us or MooTools or whatever?," 01 2009. [Online]. Available: http://stackoverflow.com/questions/176324/why-does-everyone-like-jquery-more-than-prototype-script-aculo-us-or-mootools-or. [Accessed

02 2013].

[91] N. Torn, "1. Demo "Javascript tutorial periodical live update visitor counter with Ajax and Prototype"," [Online]. Available: http://webdesign.torn.be/tutorials/javascript/prototype/live-update-visitors/#demo. [Accessed 02 2013].

[92] M. Jäger, "Das obligatorische "Hallo Welt" Beispiel," 05 2007. [Online]. Available: http://ajax.frozenfox.at/ajax_001.html. [Accessed 02 2013].

[93] jQuery Foundation, "jQuery.org: History," 2010. [Online]. Available: http://jquery.org/history/. [Accessed 02 2013].

[94] S. Narayan, "What is jQuery and How to Start using jQuery?," 02 2011. [Online]. Available: http://www.codeproject.com/Articles/157446/What-is-jQuery-and-How-to-Start-using-jQuery. [Accessed 02 2013].

[95] "jQuery user interface," [Online]. Available: http://jqueryui.com/. [Accessed 02 2013].

[96] "jquery mobile," [Online]. Available: http://jquerymobile.com/. [Accessed 02 2013].

[97] "Usage of JavaScript libraries for websites," W3Techs, 29 01 2013. [Online]. Available: http://w3techs.com/technologies/overview/javascript_library/all. [Accessed 01 2013].

[98] "jQuery: Category: Selectors," jQuery Foundation, 2013. [Online]. Available: http://api.jquery.com/category/selectors/. [Accessed 02 2013].

[99] "jQuery.get()," The jQuery Foundation, 2013. [Online]. Available:

http://api.jquery.com/jQuery.get/. [Accessed 02 2013].

[100] loyalpenguin, "Stackexchange: When to use PHP or ASP.NET?," 05 2011. [Online]. Available: http://programmers.stackexchange.com/questions/65414/when-to-use-php-or-asp-net. [Accessed 01 2013].

[101] "tizag: asp vs php," Seattle Web Design, 2008. [Online]. Available: http://www.tizag.com/aspTutorial/aspVersusPHP.php. [Accessed 01 2013].

[102] "monster," monster, 01 2013. [Online]. Available: http://jobsearch.monster.com/search/asp_5?. [Accessed 01 2013].

[103] "W3Techs - World Wide Web Technology Surveys," Q-Success, 2013. [Online]. Available: http://w3techs.com/. [Accessed 02 2013].

[104] "ASP vs PHP," [Online]. Available: http://www.aspvsphp.com/. [Accessed 01 2013].

[105] "PHP vs ASP," 11 2001. [Online]. Available: http://forum.chip.de/webentwicklung-datenbanken/php-vs-asp-241020.html. [Accessed 02 2013].

[106] H. Pires, "Webpronews.com: ASP vs. PHP," 12 2005. [Online]. Available: http://www.webpronews.com/asp-vs-php-2005-12. [Accessed 01 2013].

[107] S. Broadley, "PHP versus ASP," [Online]. Available: http://www.me-u.com/php-asp/phpvsasp.htm. [Accessed 01 2013].

[108] Zend Technologies, "zend framework 2: About," Zend Technologies, 2013. [Online]. Available: http://framework.zend.com/about/. [Accessed 02 2013].

[109] Sensio Labs, "Symfony is a PHP framework for web projects.," 2013. [Online]. Available: http://symfony.com/. [Accessed 01 2013].

[110] Yii Software LLC, "yiiframework: The Fast, Secure and Professional PHP Framework," 12 2012. [Online]. Available: http://www.yiiframework.com/. [Accessed 01 2013].

[111] Cake Software Foundation, Inc., "CakePHP makes building web applications simpler, faster and require less code.," Cake Software Foundation, Inc., 2013. [Online]. Available: http://cakephp.org/. [Accessed 01 2013].

[112] P. Brady, "PHP Framework Benchmarks: Entertaining But Ultimately Useless," 02 2010. [Online]. Available: http://blog.astrumfutura.com/2010/02/php-framework-benchmarks-entertaining-but-ultimately-useless/. [Accessed 01 2013].

[113] W3Techs, "W3Techs: Usage of server-side programming languages broken down by ranking," W3Techs, 14 02 2013. [Online]. Available: http://w3techs.com/blog/entry/fact_20121114. [Accessed 02 2013].

[114] DATACOM Buchverlag GmbH, "ITWissen.info: Rich-Client," 2013. [Online]. Available: http://www.itwissen.info/definition/lexikon/Rich-Client-rich-client.html. [Accessed 01 2013].

[115] Red Hat Middleware, LLC, "The Seam Framework - Next generation enterprise Java development," 06 2010. [Online]. Available: http://www.seamframework.org/. [Accessed 01 2013].

[116] T. Ziemer, "ziemer's Informatik: JSF Pages (CRUD) aus Entity Classes erstellen," 2013. [Online]. Available: http://www.ziemers.de/tutorials/webapp/implementierung/jsf_aus_entity_clas

ses_erstellen.html. [Accessed 01 2013].

[117] W3C, "W3C: WEB APPLICATIONS (WEBAPPS) WORKING GROUP," 2013. [Online]. Available: http://www.w3.org/2008/webapps/. [Accessed 01 2013].

[118] K. Wähner, "Einsatz und Grenzen von Java Server Faces 2.0," 09 11 2010. [Online]. Available: http://kai-waehner.de/files/Java_Server_Faces_2_0_Einsatz_und_Grenzen_Kai_Waehner.pdf. [Accessed 01 2013].

[119] "What is the need of JSF. When UI can be achieved from css html javascript jQuery?," 2011. [Online]. Available: http://stackoverflow.com/questions/4421839/what-is-the-need-of-jsf-when-ui-can-be-achieved-from-css-html-javascript-jquery. [Accessed 01 2013].

[120] "What are the main disadvantages of Java Server Faces 2.0?," 05 2010. [Online]. Available: http://stackoverflow.com/questions/4091285/jsf-adoption-and-popularity. [Accessed 02 2013].

[121] wormly, "wormly: Virtual Memory Swapping," [Online]. Available: https://www.wormly.com/help/performance-monitoring/vm-swapping. [Accessed 01 2013].

[122] R. C. Martin, "Clean Code," in *Clean Code - A Handbook of Agile Software Craftmanship*, Canada, Prentice Hall, 2009, p. 75ff.

[123] "The Ski Channel," 12 November 2011. [Online]. Available: http://www.theskichannel.com/sport/freestyle-aerial-skiing/. [Accessed Oktober 2012].

[124] "Fis Freestyle Skiing Worldcup," [Online]. Available: http://www.fisfreestyle.com/fis-info/fis-info11.html. [Accessed Oktober 2012].

[125] M. Doyle, "About.com Skiing," [Online]. Available: http://skiing.about.com/od/skiingglossary/g/freestylemoguls.htm. [Accessed Oktober 2012].

[126] "Wissenswertes.at," [Online]. Available: http://www.wissenswertes.at/index.php?id=snowboard-disziplinen. [Accessed Oktober 2012].

[127] A. Grauvogl, "netzathleten," 1 11 2011. [Online]. Available: http://www.netzathleten.de/Sportmagazin/Sports-Inside/Ski-Slopestyle-Szene-Lifestyle-meets-Olympia/670015003617961869/head. [Accessed Oktober 2012].

[128] G. Reese, R. J. Yarger and T. King, "SQL, so wie es MySQL versteht," in *Mysql - Einsatz und Programmierung*, O'Reilly Germany, 2002, p. 42.

[129] I. Teo, "PHP Master | Understanding the Factory Method Design Pattern," Dezember 2011. [Online]. Available: http://phpmaster.com/understanding-the-factory-method-design-pattern/. [Accessed Dezember 2012].

[130] C. Kunz and S. Esser, "Blacklist oder Whitelist?," in *PHP-Sicherheit: PHP/MySQL-Webanwendungen sicher programmieren*, dpunkt.verlag, 2008.

[131] M. Kofler, "The Definitive Guide to MySQL5," in *The Definitive Guide to MySQL5*, Apress, 2005, pp. 694-700.

[132] F. Trucchia and J. Romei, "Changing Class Responsibilities," in *Pro PHP*

Refactoring, Apress, 2010, pp. 107-115.

[133] "PHP Manual - JSON_DECODE," [Online]. Available: http://php.net/manual/de/function.json-decode.php. [Accessed Oktober 2012].

[134] T. Guest, "Word Aligned," 5 Juli 2007. [Online]. Available: http://wordaligned.org/articles/the-third-rule-of-program-optimisation. [Accessed Dezember 2012].

[135] J. Lengsdorf, "Common jQuery Actions and Methods," in *Pro PHP and jQuery*, Apress, 2010, pp. 25-30.

[136] J. Lengstorf, "Common Jquery Actions and Methods," in *Pro PHP and jQuery*, Apress, 2010, pp. 58-59.

[137] Dreftymac, "By User:Dreftymac, Übersetzung von User:Stf [GFDL (http://www.gnu.org/copyleft/fdl.html), CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/) or CC-BY-2.5 (http://creativecommons.org/licenses/by/2.5)], via Wikimedia Commons," 10 05 2009. [Online]. Available: http://commons.wikimedia.org/wiki/File%3ATempDeXslt015.svg. [Accessed 02 2013].

[138] E. Meyer, "A Quick Tour of CSS2," 02 1998. [Online]. Available: http://meyerweb.com/eric/articles/webrev/199802a.html. [Accessed 02 2013].

[139] J. Kyrnin, "What is the Difference Between CSS2 and CSS3: The Major Changes to CSS3," About.com, [Online]. Available: http://webdesign.about.com/od/css3/a/differences-css2-css3.htm. [Accessed 02 2013].

[140] S. Broadley, "Convert PHP to ASP," PHP / ASP Web Designers, London, [Online]. Available: http://www.me-u.com/php-asp/. [Accessed 01 2013].

[141] Kaazing Corporation, "WebSocket.org: What is WebSocket?," 2012. [Online]. Available: http://www.websocket.org/. [Accessed 02 2013].

[142] R. Gravelle, "Comet Programming: Using Ajax to Simulate Server Push," [Online]. Available: http://www.webreference.com/programming/javascript/rg28/index.html. [Accessed 02 2013].