Universität Regensburg
Fakultät für Wirtschaftswissenschaften
Professur für Wirtschaftsinformatik - Prof. Dr. Guido Schryen

Entwicklung eines Zeitplan-Generators für Leichtathletikveranstaltungen

Projektseminar

Eingereicht bei: Prof. Dr. Guido Schryen

Eingereicht am 11. Februar 2016

Eingereicht von:

Thomas Baumer, Benedikt Bruckner
E-Mail Adresse: Thomas.Baumer@stud.uni-regensburg.de, Benedikt Bruckner@stud.uni-regensburg.de
Matrikelnummer: 1721141, 1728475

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

HTML	Hypertext Markup Language
ISO	International Organization for Standardization
UTF-8	8-Bit UCS Transformation Format
API	Application programming interface
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
GUI	Graphical user interface
CSS	Cascading style sheets

Kapitel 1

Einleitung

Für die Organisation von Leichtathletikveranstaltungen bedarf es einer strukturierten und vor allem richtigen Zeitplanung, um Unannehmlichkeiten zu vermeiden bzw. einen reibungslosen und professionellen Ablauf der Veranstaltung zu gewährleisten. Momentan erhält man die Daten für die Erstellung eines Zeitplans der Veranstaltungen der LG Telis Finanz aus einer Teilnehmerliste bzw. aus einem parallel gepflegten Zeitplandokument. Dabei müssen die Informationen des jeweiligen Dokuments manuell ins Hypertext Markup Language (HTML)-Format übertragen werden, um den Zeitplan zu erstellen. Da die Teilnehmerliste mit den relevanten Daten sehr unstrukturiert ist und viele überflüssige Informationen enthält, ist der Zeitplan natürlich sehr aufwändig zu generieren. Zudem entstehen durch die redundante Datenhaltung und die manuelle Übertragung ins HTML-Format manchmal inkonsistente Zeitplandaten. Diese Gefahr besteht bei der Teilnehmerliste und im manuell gepflegten Zeitplandokument.

Im Rahmen des Projektseminars „Erstellung eines Zeitplan-Generators für Leichtathletikveranstaltungen“ soll deshalb ein Tool entwickelt werden, welches ermöglicht, dass aus der Teilnehmerliste, welche im Folgenden als HTML-Ursprungsdatei bzw. Inputfile bezeichnet wird, einer bestimmten Form automatisch ein strukturierter Zeitplan in HTML-Form erstellt wird. Das Tool erkennt alle benötigten Daten in der Ursprungsdatei, die für die Erstellung des Zeitplans notwendig sind und generiert daraus eine Tabelle. Dabei sollen das Datum der Veranstaltung, die Uhrzeit des Wettkampfbeginns, die Altersklasse (offizielle Bezeichnung) mit dem zugehörigen Geschlecht und die Wettkampfart (mit allgemeingültigen Abkürzungen) Bestandteil der gewünschten Ergebnistabelle sein. Ziel ist es für jeden Wettkampftag eine separate Tabelle zu erzeugen. Gleichzeitig soll das Datum als Überschrift für die jeweiligen Zeitpläne aufgenommen werden. Für die Ergebnistabelle selbst dient als Referenz der Zeitplan der Sparkassen Gala 2016.[1] Die Herausforderung besteht in der Korrektheit, Kompaktheit und Übersichtlichkeit der Tabelle.

Der motivierende Grund für die Umsetzung des Projektes ist die Richtigkeit und Konsistenz der Daten bei einer automatischen Generierung. Das Tool erkennt alle relevanten Daten maschinell, was einen Datenverlust ausschließt und einen vollständigen Zeitplan garantiert. Die Fehlerquelle beim manuellen Erstellen des Terminplans wird

[1] http://www.sparkassen-gala.de/zeitplan.php, abgerufen am 21.11.16

also eliminiert, da beispielsweise keine Tippfehler oder falsche bzw. fehlende Eintragungen mehr möglich sind. Dieser Punkt ist für die Leichtathleten bzw. alle Interessierten für Leichtathletikveranstaltungen von essentieller Bedeutung, da auf die im Zeitplan angegebenen Terminplaninformationen die Anreise, das Training und der Tagesablauf abgestimmt werden. Dies hätte im schlimmsten Fall zur Folge, dass der Sportler zu seinem Wettkampf nicht antreten kann, da er aufgrund einer falschen Information zu spät oder gar nicht zum Wettkampfort angereist ist. Das würde natürlich ein schlechtes Licht auf die Veranstaltung bzw. den Veranstalter werfen.

Außerdem hat die automatische Generierung eines Zeitplans für Leichtathletikveranstaltungen die positive Auswirkung, dass beispielsweise der Ersteller deutlich weniger Zeit für die Zeitplanerstellung benötigt. Es ist nicht mehr nötig mühsam alle relevanten Informationen zu identifizieren und diese anschließend in den Zeitplan zu übertragen, da dies der Zeitplan-Generator komplett übernimmt. Der Nutzer muss lediglich das gewünschte Inputfile auswählen, um einen fertigen und übersichtlichen Zeitplan zu erhalten.

Des Weiteren ermöglicht der Zeitplan-Generator eine einheitliche Darstellung von Leichtathletikzeitplänen. Alle Terminpläne für die jeweiligen Veranstaltungen werden nach dem gleichen klarem Schema und Design erzeugt, was es für den Nutzer natürlich auch leichter macht sich zurecht zu finden. Bei der Erstellung werden immer die offiziellen Altersklassen bzw. Kategorien der Leichtathletikwettbewerbe berücksichtigt. Zudem muss sich der Nutzer nicht bei jeder neuen Veranstaltung auf unterschiedliche Namenskonventionen bzw. eine andere Tabellendarstellung umstellen, da die Daten immer gleich strukturiert sind und beispielsweise keine Teilnehmerklassen zusammengefasst werden. Dies führt natürlich wieder dazu, dass unnötige Missverständnisse vermieden werden und die Benutzerfreundlichkeit erhöht wird.

Ein weiterer Vorteil bei der automatischen Erstellung eines Zeitplans ist, dass das Tool vielseitig verwendet werden kann. Als Input wird lediglich ein HTML-File benötigt, dass in einer bestimmten Form die relevanten Daten für die Terminplanerstellung enthält. Dies macht es beispielsweise auch für kleinere Vereine leicht möglich einen Terminplan zu generieren, der online verfügbar ist und den allgemeinen Standards entspricht.

Es gibt also viele gute Gründe, die für eine Entwicklung des Zeitplan-Generators für Leichtathletikveranstaltungen sprechen. Besonders die Strukturiertheit und Richtigkeit sprechen für die Verwirklichung des Projektes. Im Folgenden wird nun auf das genaue Vorgehen bei der Entwicklung des Zeitplan-Generators eingegangen.

Kapitel 2

Hauptteil

2.1 Grundlegende Überlegungen zur Durchführung des Projektes

Das Projektseminar „Erstellung eines Zeitplan-Generators für Leichtathletikveranstaltungen“ wird von Thomas Baumer und Benedikt Bruckner, im Folgenden als die Studenten bezeichnet, unter Betreuung von Gerit Wagner durchgeführt. In der Anfangsphase des Projektes stellte sich die Frage wie das Projekt umgesetzt werden soll und welche Software bzw. Programmiersprache zur Problemlösung verwendet werden sollen.

Für die Wahl einer Skriptsprache wurde zwischen den Möglichkeiten der Verwendung von JavaScript und Python diskutiert. Für beide Sprachen existieren eine Vielzahl an Dokumentationen, Bibliotheken und Tutorials. Beispielsweise kann auf die zur Programmierung benötigte Bibliothek jQuery bei JavaScript zurückgegriffen werden. Da die Studenten bereits erste Erfahrungen in JavaScript sammeln konnten, entschied man sich für JavaScript und sparte sich so eine größere Einarbeitungszeit in eine neue Skriptsprache. Mit der Verwendung von JavaScript können alle gewünschten Ziele erreicht werden.

Des Weiteren wurde die Verwendung einer Versionsverwaltungssoftware diskutiert. Da eine Versionsverwaltung viele Vorteile bietet, wie beispielsweise das Wiederherstellen von alten Zuständen des Projekts und die Protokollierung, wo jede Änderung an den Dateien mit Autor und Datum, also die ganze Versionsgeschichte, nachvollzogen werden kann, entschied man sich auch schnell und eindeutig für die Möglichkeit einer Versionsverwaltung. Außerdem ist es viel leichter, nachvollziehbarer und übersichtlicher als beispielsweise das Verschicken von einzelnen Codeteilen per E-Mail an die anderen Projektteilnehmer. Zudem bietet eine Versionsverwaltung noch die Möglichkeit Zugriffe und Entwicklungszweige zu koordinieren, was jedoch bei diesem Projekt aufgrund der Komplexität hinsichtlich der Projektteilnehmer nicht zwingend notwendig ist.[1] Als Versionsverwaltungssoftware einigte man sich schließlich auf GitHub mit der Graphical user interface (GUI) des „GitHub Desktops“.

Als Editor zur Erstellung von HTML- bzw. JavaScript-Code wurde einerseits unter

[1] Skript des Kurses Praxis des Programmierens SS16: Versionsverwaltung Folien 29ff.

Windows Notepad++ und andererseits für Mac OS die Software Brackets verwendet, da diese Open Source-Editoren sind und viele Sprachen, wie beispielsweise JavaScript und HTML unterstützen. Sie genügen also allen im Projekt erwartenden Ansprüchen. Bei der Entwicklung wurde vordergründig der Browser Google Chrome verwendet, da dieser alle verwendeten Sprachen, Frameworks und Methoden unterstützt. Darüber hinaus wurden auch die Browser Safari, Firefox und Opera für Tests miteinbezogen.

Ein weiteres Ziel ist eine fertige Desktop-App zu entwickeln. Dafür macht man sich die Software node.js in Verbindung mit dem Framework Electron zunutze. Dies bietet den großen Vorteil, dass man unabhängig vom Browser ist und beispielsweise im JavaScript-Code sich für die jeweiligen Browser umständliche Anpassungen spart. Ein weiteres Argument ist, dass nach dem Erstellen des eigentlichen Programms mit dem Editor nur noch kleine Anpassungen vorgenommen werden müssen, um ein fertiges Programm zu erhalten. Dies hat die positive Auswirkung, dass der Nutzer nicht extra die verschiedenen HTML-Dateien abspeichern muss, womit er sich möglicherweise auch nicht sehr gut auskennt. Ein eigenständiges Programm erleichtert die Bedienbarkeit und die Nutzerfreundlichkeit.

Abgesehen von der Funktionalität soll das Zeitplan-Tool auch eine ansprechende Oberfläche besitzen. Dies erhöht die Benutzerfreundlichkeit und lässt den Zeitplan bzw. die Veranstaltung auch besser erscheinen. Da der Zeitplan auch für große Leichtathletikveranstaltungen verwendet werden soll, ist die Oberflächengestaltung für die Wahrnehmung der Veranstaltung mitentscheidend. Der Nutzer erwartet eine klare und optisch ansprechende Darstellung. Um auch ein klares Design für den Zeitplan zu entwickeln, verwendet man das beliebte Cascading style sheets (CSS)-Framework Bootstrap. Bootstrap bietet eine große Menge an Gestaltungsvorlagen für Oberflächengestaltungselemente, wie beispielsweise für Buttons oder Formulare. Diese ermöglichen eine einfache und schnelle Umsetzung eines ansprechenden Designs für das Zeitplan-Tool.[2] Der Entwickler benötigt also nur ein Basiswissen in HTML und CSS, um mit Bootstrap zu arbeiten. Ein weiterer großer Vorteil ist, dass Bootstrap mit allen gängigen Browsern kompatibel ist. Zudem wird eine angepasste Darstellung auf mobilen Engeräten unterstützt.[3]

Darüber hinaus diskutierte man zu diesem frühen Stadium des Projektes bereits wie die abschließende Projektseminararbeit geschrieben werden soll. Betreuer Gerit Wagner empfahl die Verwendung des Softwarepaketes LaTex, das bei der Erstellung von größeren Abschlussarbeiten viele Vorteile gegenüber Microsoft Office Word bietet, abgesehen von dem Nachteil der Einarbeitungszeit in die neue Umgebung. Ein Argument ist, dass LaTex die Formatierung und den Inhalt voneinander trennt, indem man zu verändernde Textstellen mit Befehlen in einem Editor kennzeichnet, was ein sauberes und genau festlegbares Layout zur Folge hat.[4] Außerdem gibt es seitens des Lehrstuhls bereits eine LaTex-Vorlage für Projektseminararbeiten auf die zurückgegriffen werden kann. Somit können die strengen Anforderungen für die Formatierung bzw. Gestaltung einer

[2] https://de.wikipedia.org/wiki/Bootstrap_(Framework),abgerufenam30.11.16
[3] http://www.w3schools.com/bootstrap/bootstrap_get_started.asp , abgerufen am 30.11.16
[4] https://de.wikipedia.org/wiki/LaTeX, abgerufen am 21.11.16

umfangreichen Projektseminarabschlussarbeit durch ein sauberes Layout erzielt werden. Zudem gibt es bereits grafische Editoren die den Umgang mit LaTex vereinfachen. Im Rahmen des Projekts wurde die Software TeXworks verwendet. Des Weiteren kann man LaTex-Dokumente mit Hilfe von Git versionieren und so wieder alle Vorteile einer Versionsverwaltung nutzen.

Nachdem alle für die Problemlösung benötigten Programme, Frameworks und Skriptsprachen festgelegt wurden, wird im Folgenden die Vorgehensweise zur Durchführung des Projektes beschrieben.

2.2 Konzeptionelle Phase und Identifikation möglicher Probleme bzw. Schlüsselstellen

Dieser Unterpunkt soll einen Überblick über die allgemeine Vorgehensweise geben, die später noch detaillierter erläutert wird. Nach ersten Überlegungen mit welchen Mitteln das Projekt umgesetzt werden soll, folgten viele Gedanken über mögliche Schwierigkeiten bzw. die Herangehensweise bei der Erstellung des Zeitplan-Generators. Zuerst wurde natürlich die Ursprungsdatei genauer analysiert. Es wurde festgestellt, dass die Ursprungsdatei sehr viele irrelevante Daten beinhaltet und man jeweils nur die zwei Absätze über den Teilnehmerlisten benötigt. Parallel dazu verglich man die Zusammensetzung der Daten mit den dafür vorgesehenen Positionen in der Ergebnistabelle. Als Referenz für die Ergebnistabelle wurde wieder der Zeitplan der Sparkassen Gala 2016 verwendet. Es stellte sich heraus, dass das Auslesen der relevanten Daten eine Schwierigkeit werden könnte, da die Datensätze teilweise unterschiedliche Strukturen haben und die Informationen oft anhand mehrerer Suchkriterien identifiziert werden müssen. Des Weiteren wurde deutlich, dass die Befüllung der Tabelle eine Herausforderung werden kann, da die Wettkämpfe anhand des Datums, der Uhrzeit und der Teilnehmerklasse zugeordnet werden müssen. Bei der Umsetzung muss also ein besonderes Augenmerk auf die Identifikation der relevanten Daten und die saubere bzw. dynamische Befüllung der Ergebnistabelle gelegt werden.

Ausgehend der gemeinsamen Überlegungen im Team wurden fünf größere Schritte für die Problemlösung bzw. Erstellung des Zeitplan-Generators definiert: Der erste Schritt ist die Spezifikation des Rohtextes. In diesem Prozess wird die genau Form des Inputfiles festgehalten, was insbesondere den Inhalt und die Struktur beinhaltet. Dabei ist es von besonderer Wichtigkeit zu erkennen an welchen Positionen die wesentlichen Daten, wie Datum, Uhrzeit, Teilnehmerklasse und Wettkampfbezeichnung stehen. Der darauffolgende Schritt ist das Einlesen der Ursprungsdatei. Um mit dem Rohtext zu arbeiten muss dieser natürlich zuerst vom Programm identifiziert und eingelesen werden. Hier ist es die Aufgabe des Nutzers die gewünschte Datei, die die zuvor spezifizierte Form aufweist, auszuwählen. Der dritte Schritt ist, dass man aus der Ursprungsdatei nur die relevanten Daten herausfiltert. Hier sollen praktisch nur noch alle relevanten Absätze betrachtet werden. Anschließend müssen diese Daten im vierten Schritt gesäubert werden. Das bedeutet,

dass anhand bestimmter Suchkriterien das Datum, die Uhrzeit, die Teilnehmerklassen und die Wettkampfbezeichnung erkannt und strukturiert abgelegt werden. Im letzten Schritt wird schließlich der gewünschte Zeitplan aus den zuvor identifizierten Informationen dynamisch generiert. Für jeden Wettkampftag soll dabei eine eigene Tabelle erzeugt werden. Als Überschrift für eine Tabelle dient das Datum mit dem zugehörigen Wochentag. Der Beginn der Veranstaltung steht in der ersten Spalte unter dem Titel Zeit, wobei aufsteigend nach der Uhrzeit sortiert wird; die Altersklassen werden in der ersten Zeile ab der zweiten Zelle angeführt, wobei diese aszendierend und nach dem Geschlecht sortiert dargestellt werden und die Wettkampfbezeichnung wird schließlich in der richtigen Zelle ausgehend von Datum, Uhrzeit und Altersklasse eingetragen. Zusammenfassend werden also neben dem vorhandenen Inputfile vier HTML-Files mit JavaScript erstellt, die der Problemlösung dienen. Dies hat den Vorteil der Modularität der unterschiedlichen Funktionen und des weiteren ist es bei der Programmierung leichter umzusetzen. Zudem ist eine gute Testbarkeit der Funktionalität bzw. eine einfachere Wartung und Debugging ermöglicht. Die genaue Vorgehensweise in den einzelnen Schritten wird nun im Folgenden genauer erläutert.

2.3 Spezifikation des Rohtextes

Die erste Anforderung an den Rohtext ist, dass dieser das Dateiformat HTML besitzen soll. Dies ermöglicht eine genaue Navigation mit jQuery. Im Folgenden wird nun der Inhalt und die Struktur des Inputfiles näher spezifiziert. Grundsätzlich kann der Rohtext beliebigen HTML-Code beinhalten mit der Ausnahme des im später erläuterten Aufbau eines Paragraphen. Die für die Entwicklung verwendete Ursprungsdatei weist beispielsweise folgenden Aufbau auf. In den ersten Zeilen des HTML-Dokuments steht die Überschrift (<title>Laufnacht und Sparkassen Gala 2016</title>) und ein JavaScript-Teil, der die Positionierung beim Laden des Files festlegt (script-Tag). Daraufhin folgen verschiedene Verweise, wie die Altersklassen in Form einer Tabelle (z.B. „Männer“ und „weibliche Jugend U20“ mit der Form <td class="tdMBar»<a ... /></td>) und alle Wettkampfbezeichnungen (z.B. <a ...>100m (Vorprogramm) - Zeitläufe<a/>), die durch ein div-Tag umschlossen sind. Dieser Aufbau wird hier nicht näher betrachtet, da er für die Zeitplangenerierung keine Rolle spielt. Im Anschluss kommen die relevanten Zeilen für die Erstellung des Zeitplanes. Im listing ist hier beispielhaft der erste Paragraph in der Ursprungsdatei aufgeführt.

```
1 <p class="ev1">
2         <a name="21000001">
3                 100m (Vorprogramm), Frauen + U20 + U18 - Zeitläufe
4         </a>
5                 <br>
6         <span class="ev2">
7                 Datum: 05.06.2016  Beginn: 12:00
8         </span>
```

```
9 </p>
```

Im Testfile ergibt sich folgender Klartext:

Zeile 1: „100m (Vorprogramm), Frauen + U20 + U18 – Zeitläufe“

Zeile 2: „Datum: 05.05.2016 Beginn: 12:00“

Diese zwei Zeilen sind DOM-Elemente der Form ... . Es handelt sich hier also um Paragraphen und man kann alle relevanten Daten mit Hilfe dieser Form identifizieren. Der Paragraph ist im Allgemeinen also folgendermaßen aufgebaut: Zeile 1 beinhaltet die Teilnehmerklassen (z.B. „Frauen + U20 + U18“) ungefähr in der Mitte der Zeile. im String gekennzeichnet folgt auf die ersten Zeichen und ein Komma (xxxx,) die Teilnehmerklasse. Auf die Teilnehmerklasse folgt ein Bindestrich und die nachfolgenden Zeichen (-xxxx). Die Teilnehmerklassen werden also initiiert durch ein Komma und beendet durch einen Bindestrich. Zudem enthalten alle Teilnehmerklassen die Zeichenfolgen „weiblich“, „weibliche“ oder „Frauen“ bzw. „männlich“, „männliche“ oder „Männer“. Außerdem ist es möglich, dass in diesem Abschnitt mehrere Teilnehmerklassen, abgetrennt durch ein „+“, enthalten sind. Auf das Pluszeichen würden dann ein „U“ und zwei Zahlen folgen. Ein weiterer Bestandteil der ersten Zeile ist gegebenenfalls der Inhalt der runden Klammern. Dort befinden sich Zusatzinformationen, die entfernt werden, wenn es sich um „Hauptprogramm“, „Gala“, „Vorprogramm“ oder „Laufnacht“ handeln sollte. Ansonsten bleibt der Inhalt der Klammern an dieser Stelle stehen. Es gibt aber auch den Fall, dass keine Zusatzinformationen in runden Klammern enthalten sind. Den Rest der Zeichenkette (nach Entfernung der Teilnehmerklasse und gegebenenfalls der Klammern) bildet die Disziplin. Die Disziplin setzt sich also aus den ersten Zeichen bis zur Klammer und den letzten Zeichen nach dem Bindestrich zusammen. Im Beispiel entsteht die Disziplin „100m Zeitläufe“.

Die zweite Zeile enthält das Datum und die Uhrzeit. Der erste Eintrag in der zweiten Zeile ist „Datum“ gefolgt vom Datum selbst in der Form dd.mm.yyyy. Der zweite Eintrag in der zweiten Zeile ist „Beginn“ gefolgt von der Uhrzeit des Wettkampfstarts mit dem vierundzwanzig Studenformat hh:mm.

Abgesehen davon kommt nach den Paragraphen in der verwendeten Ursprungsdatei immer eine Tabelle mit der Startnummer, dem Namen, dem Jahrgang, der Nationalität, dem Verein, der Saisonbestleistung und der persönlichen Bestleistung (<th>-Tag) , sowie den verschiedenen Einträgen der Wettkampfteilnehmer (<td>-Tag). Dies ist jedoch für die Zeitplangenerierung nicht weiter relevant.

Zusammenfassend muss das Ausgangsfile also im HTML-Format vorliegen und DOM-Elemente mit der Form ... enthalten, die als Inhalt die relevanten Daten haben. Zudem müssen die Paragraphen den wie oben beschriebenen Aufbau aufweisen, damit alle Daten im Anschluss richtig verarbeitet werden können. Nachdem der Rohtext genauer spezifiziert wurde, wird die Funktionsweise des File-Uploaders im Folgenden genauer erklärt.

2.4 Funktionalität des File-Uploaders

Nachdem das Inputfile genauer spezifiziert wurde, wird nun die Funktionsweise des File-Uploaders näher erklärt. Die Vorbedingung für den File-Uploader ist, dass der Rohtext in der zuvor definierten Form vorliegt.

Im ersten Schritt wird nun der HTML-Code erläutert. Dieser setzt sich aus dem HTML-Kopf (HEAD) und dem HTML-Körper (BODY) zusammen, die jeweils auch durch die gleichnamigen Tags eingeleitet und beendet werden. Da der File-Uploader ein Teil der Benutzeroberfläche ist, soll diese auch ansprechend mit Bootstrap bzw. CSS gestaltet werden. Zu Beginn des HEADs müssen für die Verwendung von Bootstrap drei Meta-Tags (<meta />) deklariert werden. Dabei gibt das erste Meta-Tag den zu verwendenden Zeichensatz 8-Bit UCS Transformation Format (UTF-8) an. Die Verwendung von UTF-8 als Zeichensatz hat den Vorteil, dass alle Zeichen (auch Fremdwörter und Sonderzeichen) beliebig verwendet werden können.[5] Des weiteren ist UTF-8 beispielsweise von der Internet Engineering Task Force (IETF) und der International Organization for Standardization (ISO) als Norm definiert worden, was die Nutzung begünstigt.[6] Das zweite Meta-Tag muss ebenfalls standardmäßig definiert werden, da so eine bestmögliche Darstellung im Internet Explorer, auch in den älteren Versionen, gewährleistet werden kann.[7] Die Metainformation mit dem Namen viewport stellt sicher, dass die Website auch auf mobilen Geräten richtig dargestellt wird. Dies ist ein sehr wichtiger Faktor, da heutzutage eine Vielzahl der Nutzer den mobilen Browser auf einem Smartphone verwenden und sich auf diesem Weg den Zeitplan anschauen wollen. Durch den Ausdruck width=device-width wird eine genaue Anpassung der Website an die Bildschirmgröße des mobile Endgeräts sicher gestellt. Des weiteren wurde hier die Möglichkeit des Zoomens nicht ausgeschlossen, da die Darstellung des Zeitplans auf einem mobilen Endgerät eventuell relativ klein ausfällt und der Nutzer die Informationen nur schwer erkennen kann. Das Zoom-Level wird beim ersten Laden der Website (initial-scale) mit dem Browser auf 1 gesetzt.[8] Im Anschluss wurde im HEAD noch der Titel des Web-Dokuments und zur Einbindung von Bootstrap die verwendete (minimierte) Bootstrap-CSS-Version definiert.

Im BODY wird zunächst die Überschrift ("page-header") festgelegt. Die Überschrift soll dabei separat, durch eine horizontale Linie unter der Überschrift, von den anderen Inhalten des File-Uploaders dargestellt werden. Dies wurde mit dem div-Element in Verbindung mit der Klasse .page-header realisiert.[9] Die Überschrift soll der Nutzerhinweis „Bitte die Teilnehmerliste einlesen“ sein. Neben der Überschrift muss im File-Uploader noch ein Button plaziert werden, wo der User das gewünschte File auswählen kann. Da der Button einer der zentralen Bestandteile dieses Web-Dokuments ist, entschied man sich zuerst einen gesonderten grauen Bereich unterhalb der Trennlinie zur Überschrift zu definieren, wo später der Button plaziert werden soll. Dies wurde mit der Bootstrap Wells-

[5] https://wiki.selfhtml.org/wiki/Zeichenkodierung, aberufen am 23.11.16
[6] https://de.wikipedia.org/wiki/UTF-8, abgerufen am 23.11.16
[7] http://v4-alpha.getbootstrap.com/getting-started/browsers-devices/, aberufen am 02.12.16
[8] http://www.w3schools.com/bootstrap/bootstrap_get_started.asp, abgerufen am 02.12.16
[9] http://www.w3schools.com/bootstrap/bootstrap_jumbotron_header.asp, abgerufen am 02.12.16

Klasse umgesetzt, die um ein Element einen grauen Bereich legt.[10] Für die Darstellung des Buttons selbst entschied man sich für ein Design, das an die verwendeten Farben der LG Telis Finanz-Website angelehnt sein soll. (vgl. http://lg-telis-finanz.de) Dies soll eine konsistente Darstellung bei diesen Leichtathletikveranstaltungen unterstreichen. Bei den verschiedenen Button Styles von Bootstrap wählte man deshalb den Info-Button (.btn-info), der hellblau gefüllt ist und einen weißen Schriftzug hat. Der Schriftzug des Buttons soll der Nutzerhinweis „Datei hochladen“ sein. Außerdem enthält der Button noch die Funktion auf einen Click zu reagieren. Die ausgelöste Methode clickedButton() wird im JavaScript-Teil definiert. Sie bewirkt, dass nach dem Click auf den Button der Click auf das Inputfile ausgeführt wird, das zuvor unsichtbar bzw. versteckt war. Außerdem wurde im BODY noch der Input bestimmt, der vom Typ eines Files sein soll. Der Input wird am Anfang unter Verwendung von CSS ausgeblendet (style="display: none"), da er erst angezeigt werden soll, wenn über den Button eine Datei ausgewählt wurde (click). Dies ist eine saubere Lösung im Vergleich zu einschlägigen Vorschlägen aus dem Netz, wo der Fileinput vor dem Auslösen durch einen Click mit einem Bild verdeckt wurde. Darüber hinaus wurde die Möglichkeit für den Nutzer das File mit Hilfe eines Formulars auszuwählen integriert (form-Tag). Formulare in HTML dienen der Eingabe von Informationen durch den Benutzer. Zudem wurde im HTML-Körper noch die Bildschirmausgabe (displayArea) deklariert, die eventuelle Ausgaben anzeigt. Am Ende des BODYs wurden noch die Skripte, die für Bootstrap benötigt werden deklariert. Diese sind jQuery, das für Bootstrap JavaScript-Plugins benötigt wird, und das kompilierte JavaScript.[11]

Im JavaScript-Code wurden die restlichen Funktionalitäten des File-Uploaders festgelegt. JavaScript ist eine Skriptsprache, die HTML um Funktionalitäten erweitert und dynamische Informationen im Web realisiert. Das Skript wird in HTML-Dokumente mit dem script-Tag eingebettet. Dies ermöglicht beispielsweise, dass Benutzerinteraktionen ausgewertet bzw. Inhalte erzeugt oder verändert werden können. Diese Eigenschaften werden hier benötigt. Es wurden zuerst Variablen definiert, die die relevanten DOM-Elemente „fileInput“ und „displayArea“ abspeichern, um diese im JavaScript-Code verändern zu können bzw. um auf diese zugreifen zu können. Im Folgenden wurde der User-Input als Event definiert. Dies ermöglicht, dass man auf eine Dateiauswahl des Nutzers im Select-Auswahlmenü reagieren kann. Hat der Nutzer also eine Auswahl getroffen bzw. liegt eine Veränderung beim Inputfile vor (change), wird das Event ausgelöst. Bei der Auslösung des Events müssen zunächst einige grundlegenden Dinge zur Ausführung des JavaScript-Codes festgelegt werden. Der Code arbeitet mit dem zuerst ausgewählten File (erste Stelle im Array) und prüft anschließend, ob das File den zuvor definierte Typ HTML aufweist. Wird an dieser Stelle erkannt, dass ein anderes Format vorliegt, wird das Event abgebrochen und es wird dem Nutzer eine Fehlermeldung („File not supported!“) im displayArea ausgegeben. Handelt es sich um ein HTML-File, ermöglicht die File Reader-Application programming interface (API) das Auslesen des Textes. Hier

[10] http://www.w3schools.com/Bootstrap/bootstrap_wells.asp, abgerufen am 02.12.16
[11] http://holdirbootstrap.de/los-gehts/, abgerufen am 02.12.16

wird zur Fehlervermeidung eine Verwendung eines falschen Texttyps vermieden. Um einen korrekten deutschen Text als Ergebnis zu erhalten, muss das File nach ISO-8859-1 enkodiert werden. Dies vermeidet beispielsweise, dass im Web-Dokument Umlaute nicht richtig dargestellt werden können. Sobald das File fertig geladen ist, wird der Inhalt des Eingangsfiles in der Variable „rawText“ gespeichert. Hier macht man sich die Methoden onload und result der File Reader-API zunutze. Onload erkennt, wann das File komplett eingelesen wurde und result gibt den Inhalt des Files zurück.[12] Darüber hinaus soll das Ergebnis des Einlesens, also die zuvor definierte Variable rawText, im sessionStorage abgespeichert werden. Die Datenspeicherung wird also an die aktuelle Browsersession gebunden, d.h. die Daten bleiben deshalb auch nur so lange gespeichert bis die Sitzung geschlossen wurde. Dies ermöglicht auch ein einfacheres Debuggen, da beispielsweise die fertige Tabelle selbst noch auf den Rohtext zugreifen könnte. Dies ist ein großer Vorteil gegenüber dem Versenden der Daten mit dem form-Tag. Zudem können andere Fenster im Browser die Daten während der Sitzung nutzen. Nachdem dies geschehen ist wird der Text aus dem sessionStorage angezeigt. Diese beiden Funktionalitäten wurden in den Methoden showText(area) bzw. saveRawText(rawText) ausgelagert.

Nach dem Abschluss der Programmierphase bzw. des Testens wurde der Submit-Button und die Anzeige des eingelesenen und abgespeicherten Files im Code auskommentiert bzw. auf versteckt (hidden) gesetzt. Dies hat den Vorteil, dass man bei der Ausführung des kompletten Codes zur Erstellung der Tabelle durch das automatische Weiterleiten zum ParagraphHandler Zeit spart. Zudem ist es für den Nutzer irrelevant, dass er das eingelesene File mit den unstrukturierten Daten angezeigt bekommt. Bei Bedarf kann das Auskommentieren natürlich wieder rückgängig gemacht werden, was das Debugging bzw. die Erweiterung des Tools vereinfachen würde.

Ist also der Rohtext (rawText) im sessionStorage, leitet das Programm automatisch (Auto-Submit) zum nächsten Schritt bei der Generierung einer Tabelle, dem Paragraph-Handler, weiter. Beim Prozess der Erstellung der Ergebnistabelle wird im Folgenden die Funktionsweise des ParagraphHandlers detaillierter erklärt.

2.5 Paragraph Handler: Herausfiltern der Paragraphen aus dem Rohtext

In diesem Schritt ist es das Ziel aus dem im sessionStorage gespeicherten Rohtext die Paragraphen, die für die Zeitplanerstellung benötigt werden, zu erhalten. Darüber hinaus sollen die Paragraphen gleich unterteilt nach erster und zweiter Zeile in einem Array abgespeichert werden.

Dabei wurden zunächst im HEAD der Titel des HTML-Dokuments („Paragraph Handler!“) und im BODY die Weiterleitung zum HTML-Dokument „Cleaner“ deklariert. Zudem wurden in der Implementierungsphase auch wieder ein Submit-Button und ein displayArea definiert, um die Funktionalität zu testen bzw. Fehler zu beheben. Im An-

[12] https://www.w3.org/TR/FileAPI/, abgerufen am 24.11.16

schluss wurden die verwendeten Skripte definiert. Da man nun alle DOM-Elemente der Form eines Paragraphen (...) aus der im sessionStorage abgespeicherten Ursprungsdatei herausfiltern will, wird bei der Programmierung das Framework jQuery verwendet. jQuery bietet unter anderem den Vorteil, dass es eine einfache und schnelle Möglichkeit bietet DOM-Elemente mit CSS-artigen Selektoren auszuwählen und zu bearbeiten. jQuery ermöglicht also DOM-Abfragen mit einer einfachen Syntax und bietet zudem Methoden, die wichtige Aufgaben mit sehr wenig Code erledigen.[13] Bei der Einbindung von jQuery in den Paragraph Handler hat man sich bei der Angabe der Quelle (src) für die Google-API entschieden. Dies hat den großen Vorteil, dass jQuery nicht extra geladen werden muss, wenn vorher Websites jQeury ebenfalls über die Google-API eingebunden haben. Dies macht sich natürlich auch in der Performance bemerkbar. Des weiteren ist es komfortabler als die Einbindung über den lokalen Pfad, wo man jQuery nach dem Herunterladen abgespeichert hat. Zudem wird so gewährleistet, dass immer die aktuellste jQuery-Version verwendet wird.[14] Darüber hinaus muss jQuery als Skript für Electron und Node.js angegeben werden.

Die eigentliche Funktionalität des Paragraph Handlers wird im JavaScript-Code umgesetzt. Um mit der Ursprungsdatei arbeiten zu können, muss diese zunächst aus dem sessionStorage geladen werden. Dies geschieht in der Methode getRawText(). Anschließend wird der rawText im sessionStorage gespeichert, was in der Methode saveParagraphText(paragraphText) umgesetzt ist. Nach dem Speichern soll der Rohtext angezeigt werden, damit man mit jQuery diesen durchlaufen und die Tags erkennen bzw. finden kann. jQuery soll jetzt alle Paragraphen mit der Klasse „ev1“ finden und diese in der Variable „$html“ speichern. Mit dem Ausdruck „$('p.ev1')“ werden nun alle Elemente mit dieser Eigenschaft zurückgegeben. Anschießend werden die Elemente in der jQuery-Auswahl mit der Methode clone() kopiert.[15] Die Methode text() gibt nur den Textinhalt ohne Tags zurück. Nachdem die relevanten Informationen in der Variable $html gespeichert wurden, kann der Inhalt der Seite bzw. der angezeigte Rohtext aus dem displayArea gelöscht werden. Dies wird durch das Setzen eines leeren Strings mit Hilfe der Eigenschaft innerHTML, die den Inhalt eines HTML-Elements speichert, realisiert.[16] Anschließend soll der Textinhalt der Paragraphen mit der Klasse ev1 der neue Inhalt der Seite sein. Dies wird mit der jQuery-Methode prepend(prepended text) ermöglicht, die den Inhalt ($html) am Beginn der selektierten Elemente (pre) einfügt.[17] Da nun der Textinhalt der Paragraphen im displayArea angezeigt wird, kann der Inhalt nun mit der saveParagraphText(paragraphText)-Methode im sessionStorage abgespeichert werden.

Im nächsten Schritt im Paragraph Handler sollen die Paragraphen noch strukturiert nach erster und zweiter Zeile als Array abgespeichert werden. Dies erleichtert das weitere Vorgehen bei der Generierung des Zeitplans. Dafür wurde zunächst ein Konstruktor für einen strukturierten Paragraphen (StrucParagraph(firstLine, secondLine)) deklariert,

[13] JavaScript & jQuery - Interaktive Websites entwickeln, von Jon Duckett S.294ff
[14] http://www.html-seminar.de/jquery-tutorial.htm, abgerufen am 27.11.16
[15] http://www.w3schools.com/jquery/html_clone.asp, abgerufen am 27.11.16
[16] https://wiki.selfhtml.org/wiki/JavaScript/DOM/Element/innerHTML, abgerufen am 27.11.16
[17] http://www.w3schools.com/jquery/html_prepend.asp, abgerufen am 27.11.16

wo die Paragraphen abgelegt werden können. Die Verwendung eines Konstruktors ermöglicht eine bessere und klare Verwendung im Code. Um ein Array aus Paragraph-Objekten (StrucParagraph) zu speichern, benötigt man zuerst den im sessionStorage gespeicherten unstrukturierten Textinhalt der Paragraphen. Mit dem Aufruf „sessionStorage.getItem('paragraphText')“ werden die Daten aus dem sessionStorage abgefragt und in der Variable „text“ gespeichert. Des weiteren muss das Array selbst definiert werden. Zudem werden drei Variablen deklariert, die die Anfangs-, Mittel- bzw. Endposition des Paragraphen sein sollen. Diese Variablen sollen bei der Aufteilung des Textes in die zwei Zeilen helfen. Bei der Initialisierung werden das Array und die Variablen zunächst auf leer bzw. 0 gesetzt. Darüber hinaus wurde die Variable „patternTimeLength“ angelegt, die die Zeichenlänge der Uhrzeit angeben soll. Da sich die Uhrzeit immer aus zwei Stellen Stundenanzeige, einem Doppelpunkt und zwei Stellen Minutenanzeige zusammensetzt, ergibt sich in Summe eine Zeichenlänge von fünf. Deshalb wurde die Variable bei der Initialiserung auf fünf gesetzt.

Um den ganzen Text aus dem sessionStorage zu durchlaufen, benötigt man eine while-Schleife, die solange läuft, bis alle Daten erfasst und im Array abgespeichert wurden. Dabei soll die Schleife genau einmal für jeden Paragraphen durchlaufen werden. Es muss also zuerst ein Kriterium herausgefunden werden, um das Ende eines Paragraphen zu bestimmen. Da die Uhrzeit immer an letzter Stelle in den Paragraphen steht, wird die Position nach der Uhrzeit als Ende des aktuell betrachteten Paragraphen festgelegt. Nun muss eine Möglichkeit gefunden werden, damit das Programm diese Stelle finden kann. Da die Startzeit immer fünf Zeichen hat und immer durch das Stichwort „Beginn:“ eingeleitet wird, könnte man nach dem Stichwort suchen und die restlichen Zeichen aufaddieren, um die gewünschte Position zu erhalten. Eine weitere Möglichkeit ist, dass man die Position der Uhrzeit mit Hilfe eines regulären Ausdrucks identifiziert. Dies hat den Vorteil, dass auch wirklich sichergestellt werden kann, dass eine valide Uhrzeit im Paragraphen gefunden wird. Der reguläre Ausdruck schreibt die Form vor, die die Uhrzeit aufweisen soll. Der gesamte reguläre Ausdruck setzt sich folgendermaßen zusammen:

```
([01]\d|2[0-3]):[0-5]\d
```

Die Uhrzeit soll mit einer 0 oder 1 gefolgt von einer Zahl (0-9), im regulären Ausdruck als d für digit dargestellt, bzw. mit einer 2 gefolgt von den Zahlen 0, 1, 2, oder 3 beginnen. Dies soll die Stunden der Uhrzeit widerspiegeln. Nach den Stunden soll in der Uhrzeit der Doppelpunkt enthalten sein. Da die Minutenanzeige höchstens 59 anzeigen kann, erlaubt der reguläre Ausdruck an der ersten Stelle der Minutenanzeige nur eine Zahl zwischen 0 und 5. Die letzte Stelle der Uhrzeit darf eine Zahl zwischen 0 und 9 sein (digit).

Um im Paragraphen den Mittel- und Endpunkt zu finden und zu erhalten, verwendete man die search- bzw. slice-Methode. Alternativ wäre hier auch die Verwendung von jQuery möglich gewesen. Diese Methoden ermöglichen aber ein schnelles und einfaches Erhalten der gewünschten Positionen im Paragraphen, weshalb man sich für die Verwendung dieser Methoden im Code entschied. Mit Hilfe der Methode search wird nun nach der im regulären Ausdruck definierten Form im Text gesucht und die Position der

Übereinstimmung zurückgegeben.[18] Da die Position vor der Position der Uhrzeit geliefert wird muss nun noch die patternTimeLength dazu addiert werden, um das Ende eines Paragraphen zu erhalten. Diese Position wird dann in der Variable endSliceP abgespeichert. Mit Hilfe dieser Variable kann also genau erkannt werden, wann der Paragraph endet. Der Paragraph wird nun mit Hilfe der slice-Methode aus dem ganzen Text abgeschnitten und in die Variable storageOneParagraph geschrieben. Dabei braucht die slice-Methode als Parameterwerte einen Start- und Endpunkt, wobei der Endpunkt nicht mehr Teil des Ergebnisstrings ist.[19]

Um die Paragraphen nun in die zwei Zeilen zu unterteilen muss das Ende der ersten Zeile bzw. der Anfang der zweiten Zeile erkannt werden. Als Kriterium wurde hier das Stichwort „Datum“ gewählt, da dies bei jedem Paragraphen am Anfang der zweiten Zeile enthalten ist und nicht Bestandteil der ersten Zeile ist. Um das Stichwort „Datum“ im Paragraphen zu finden nutzt man wieder die search-Methode, die die Position des Wortes zurückgibt. Die Position wird in der Variable middleSliceLine (Mittelstück des Paragraphen) gespeichert. Für die Aufteilung des Paragraphen an dieser Stelle nutzt man wieder die slice-Methode. Die erste Zeile geht von der Stelle 0 (Anfangsposition des Paragraphen bzw. startSliceP) des Paragraphen bis zur middleSliceLine-1. Hier muss man die eine Position noch abziehen um von der Stelle des Datums in die erste Zeile zu kommen. Die zweite Zeile hat als Startpunkt die middleSliceLine und als Endpunkt die endSliceP. Die entstandenen Strings für die erste und zweite Zeile speichert man dabei gleich in den Variablen storageFirstLine bzw. storageSecondLine ab. Im Anschluss muss man nun noch die beiden Zeilen in einen StrucParagraph umwandeln und diesen zum Array hinzufügen, was mit der push-Methode umgesetzt wird. Im Array wird also der Paragraph in der Form wie er angezeigt wurde zwischengespeichert. Jetzt wird mit dem Aufruf text=text.slice(endSliceP) noch sichergestellt, dass der bereits abgearbeitete Paragraph aus dem Text weggeschnitten wird und im nächsten Durchlauf der darauffolgende Paragraph betrachtet wird.

Diese Anweisungen werden solange für jeden Paragraphen durchlaufen, bis alle Paragraphen mit der gewünschten Unterteilung in erster und zweiter Zeile zum Array hinzugefügt wurden.

Im letzten Schritt müssen die Paragraphen bzw. das Array noch gespeichert werden. Das Array wird als JavaScript Object Notation (JSON) im sessionStorage abgespeichert, um Daten zwischen den HTML-Dokumenten austauschen zu können. JSON hat den Vorteil, dass das Format kompakter ist als HTML/XML.[20] Zudem ermöglicht die Methode stringify(value) ein simples Parsen zu einem String. Das Array wird also als String in der Variable arrayJSON abgespeichert.

Nachdem das Ziel die relevanten Daten aus der Ursprungsdatei zu erhalten und die Unterteilung des Paragraphen in erste und zweite Zeile erledigt wurden, müssen die Veranstaltungsdaten im folgenden Schritt extrahiert werden. Dies wird in dem HTML-

[18] http://www.w3schools.com/jsref/jsref_search.asp, abgerufen am 27.11.16
[19] http://www.w3schools.com/jsref/jsref_slice_array.asp, abgerufen am 27.11.16
[20] JavaScript & jQuery - Interaktive Websites entwickeln, von Jon Duckett S.374

Dokument „Cleaner“ umgesetzt. Wie beim File Uploader wurde nach der Implementierungsphase das Anzeigen des Textes im displayArea bzw. der Submit-Button, um zum nächsten HTML-Dokument zu kommen auskommentiert. Es wurde wieder ein automatischer Submit integriert, der zum nächsten HTML-Dokument weiterleitet.

2.6 Cleaner: Strukturiertes Ablegen der Informationen aus den Paragraphen als Event

Nachdem die Paragraphen strukturiert nach der ersten und zweiten Zeile im sessionStorage abgelegt wurden, müssen in diesem Schritt das Datum, die Uhrzeit, die Wettkampfbezeichnung und die Wettkampfklasse in den Paragraphen identifiziert und strukturiert abgelegt werden. Zuerst wurde im HEAD wieder der Titel des Web-Dokuments und im BODY die Weiterleitung zu „GenerateTable.html“ bzw. das displayArea deklariert.

Im Folgenden wird nun der JavaScript-Code erläutert. Um mit den Paragraphen zu arbeiten, muss das mit JSON im sessionStorage abgespeicherte Array (als String) zunächst wieder initialisiert werden. Dies wird mit der Methode getParagraphesText() realisiert, die die Paragraphen als String zurückgibt. Da man für die Problemlösung das Paragraph-Objekt braucht, muss der String wieder zu einem Array aus den Objekten „StrucParagraph“ geparst werden.

Im Anschluss sollen dann für jedes StrucParagraph-Objekt die relevanten Daten identifiziert und abgespeichert werden. Da ein StrucParagraph-Objekt alle Daten für einen Wettkampf enthält, werden die Daten eines Paragraphen zusammengefasst als ein Event abgespeichert. Dies hat den Vorteil, dass die Daten sauber und strukturiert abgelegt werden können. Die Definierung eines Events ermöglicht so später eine einfachere Erstellung der Tabelle. Der Datentyp Event muss also erzeugt werden. Er setzt sich aus dem Datum der Veranstaltung (date), dem Veranstaltungsbeginn (time), der Wettkampfklasse bzw. Kategorie (category) und der Wettkampfbezeichnung (discipline) zusammen. Dabei enthalten die Variablen alle relevanten Informationen zur Erstellung eines Zeitplans. Die Variablen sind vom Typ String. Um für die in einem Array gehaltenen StrucParagraph-Objekte jeweils ein Event zu erhalten, muss ebenfalls ein Event-Array definiert werden.

Um nun das Array aus StrucParagraph-Objekten zu durchlaufen, wird wieder eine for-Schleife benötigt. Dies ermöglicht, dass eine Reihe an Schritten für jeden einzelnen Paragraphen auf einmal umgesetzt werden kann. Diese Methodik erscheint zwar unübersichtlich, aber sie erlaubt gleichzeitig eine einfache Identifizierung der Variablen für ein Event.

Am Beginn der for-Schleife nutzt man den Vorteil, dass man im StrucParagraph-Objekt die erste und zweite Zeile eines Paragraphen getrennt abgespeichert hat. Dies ermöglicht in diesem Schritt eine einfachere Identifizierung der Daten. Zuerst werden also Variablen definiert, die die erste und zweite Zeile des zu dem jeweiligen Zeitpunkt betrachteten Paragraphen widerspiegeln. Anschließend müssen die Daten, die in den jeweiligen Zeilen stehen identifiziert werden.

Im Code wurde mit der Aufarbeitung der zweiten Zeile begonnen. Die zweite Zeile enthält am Anfang die Information des Datums der Veranstaltung und ungefähr ab der Mitte der Zeile steht der Veranstaltungsbeginn. Die Daten in der zweiten Zeile werden anhand von regulären Ausdrücken identifiziert. Dies ermöglicht, dass valide Informationen als Datum bzw. Veranstaltungsbeginn in der Zeile gefunden werden.
Der reguläre Ausdruck für das Datum setzt sich folgendermaßen zusammen:

```
(3[01]|[0-2]\d)[.](0\d|1[0-2])[.]20\d\d
```

Der reguläre Ausdruck setzt sich aus der Angabe des Tages, des Monats und des Jahres zusammen. Diese werden durch einen Punkt voneinander getrennt. Ein Tag beginnt mit einer 3 gefolgt von einer 0 oder einer 1. Die andere Möglichkeit ist, dass sich ein Tag mit einer 0, 1 oder 2 gefolgt von einer Zahl zwischen 0 und 9 (d für digit) zusammensetzt. Ein Monat besteht entweder aus einer 0 gefolgt von einem digit oder aus einer 1 gefolgt von einer 0, 1 oder 2. Somit sind alle Fälle vom ersten Monat Januar bis zum letzten Monat Dezember abgedeckt. Zum Schluss muss noch die Form des Jahres definiert werden. Die Jahreszahl beginnt mit eine 2 und einer 0 gefolgt von zwei Zahlen zwischen 0 und 9. In diesem regulären Ausdruck wird nur die Validität eines Datums und nicht die Korrektheit überprüft, da der reguläre Ausdruck nur zur Identifizierung eines syntaktisch richtigen Datums dient. D.h. Sonderfälle wie der Februar, der keine 3 an erster Stelle bei der Angabe des Tages haben kann, werden nicht auf Richtigkeit geprüft. Um den regulären Ausdruck zu finden, wird die Methode search verwendet, die die Anfangsposition des regulären Ausdrucks zurückgibt. Das Ergebnis wird in der Variable dateIndex gespeichert. Um auch die Endposition des Datums zu erhalten, hat man die Variable patternDateLength mit dem Wert 10 definiert. Der Wert wurde auf 10 gesetzt, da die Anzahl der Zeichen des Datums der Form (dd.mm.yyyy) in Summe immer 10 ergibt. Das Datum besteht also aus jeweils zwei Zeichen für die Angabe des Tages und des Monats, vier Zeichen für die Darstellung der Jahreszahl und zwei Punkten, um die Werte voneinander zu trennen. Mit Hilfe dieser Variablen kann nun das Datum in der zweiten Zeile erkannt werden. Mit der slice-Methode wird nun das komplette Datum zurückgegeben und in der Variable date gespeichert.

Neben dem Datum muss noch die Uhrzeit in der zweiten Zeile identifiziert werden. Dies wird mit dem gleichen regulären Ausdruck umgesetzt, der bereits im Punkt 2.5 genauer erläutert wurde. Mit der Kombination aus search- und slice-Methode erhält man wieder das gewünschte Ergebnis (hier: die Uhrzeit) geliefert.

Jetzt müssen die Daten der ersten Zeile noch identifiziert werden. Da in der ersten Zeile in Klammern einerseits relevante und andererseits unwichtige Zusatzinformationen enthalten sein können, müssen in diesem Schritt die relevanten Informationen erkannt werden. Zusatzinformationen sind die Begriffe „Gala“, „Laufnacht“, „Vorprogramm“ und „Hauptprogramm“. Sind diese Begriffe enthalten braucht der Inhalt der Klammern nicht weiter betrachtet werden. Die Zusatzinformationen wurden in dem Array cutOutOf-Brackets abgespeichert. Um diesen Fall im Code abzubilden, wurde zuerst eine if-Schleife definiert. Diese wird nur durchlaufen, wenn Klammern enthalten sind. Sind Klammern

vorhanden, wird mit Hilfe der split-Methode die Zeile in drei Teile zerlegt.[21] Der erste Teil (brokenFirstLineBrackets[0]) erstreckt sich vom Beginn der ersten Zeile bis zur öffnenden runden Klammer, der zweite Teil (brokenFirstLineBracketsHelper[0]) ist der Inhalt der runden Klammern und der dritte Teil (brokenFirstLineBracketsHelper[1]) ist der Text nach der schließenden runden Klammer. Nun wurde der Inhalt der Klammern isoliert und es kann überprüft werden, ob eine nicht relevante Zusatzinformation enthalten ist. Dazu muss das Array cutOutOfBrackets durchlaufen werden. Ist eine Zusatzinformation vorhanden, wird diese gelöscht. Nachdem die Fälle keiner und einer irrelevanten Zusatzinformation abgedeckt wurden, wird nun für die weitere Identifizierung die erste Zeile ohne Klammern und ohne den Inhalt der Klammern zusammengesetzt. Dies ist mit den zuvor definierten Variablen brokenFirstLineBrackets[0] und brokenFirstLineBracketsHelper[1] leicht umsetzbar.

Im nächsten Schritt soll die Disziplin in der ersten Zeile identifiziert werden. Die Disziplin setzt sich aus mindestens zwei Teilen der ersten Zeile zusammen. Der erste Teil ist vom Beginn der Zeile bis zum Komma (Klammern und Inhalt der Klammern sind nicht mehr vorhanden) und der zweite Teil beginnt nach dem Bindestrich. Darüber hinaus kann der Inhalt der Klammern, wenn diese vorhanden und keine Zusatzinformation enthalten ist, ein Teil der Disziplin sein. Zuerst speichert man die einzelnen Teile wieder in Variablen ab. Der erste Teil der ersten Zeile (bis zum Komma) wird in der Variable brokenFirstLineDiscipline[0] abgespeichert. Der mittlere Teil (brokenFirstLineDisciplineHelper[0]) erstreckt sich vom Komma bis zum Bindestrich und der dritte Teil (brokenFirstLineDisciplineHelper[1]) beginnt nach dem Bindestrich. Ist die Variable, wo der Klammerninhalt abgespeichert wurde leer, kann die Disziplin aus dem ersten und dritten Teil bereits zusammengesetzt werden. Ist ein Wert vorhanden, wird der Inhalt der Klammern wieder in Klammern zwischen den beiden Teilen für die Bildung der Disziplin ergänzt. Das Ergebnis wird in der Variable discipline gespeichert. Nachdem die Disziplin erkannt wurde, sollen gewisse Schlüsselwörter davon durch die allgemeingültigen Abkürzungen ersetzt werden. Dies hat den Vorteil, dass der Zeitplan bei der Erstellung kompakter und übersichtlicher wird. Die Wörter „Zeitläufe“, „Hindernis“, „Hürden“ und „Vorläufe“ sollen dabei jeweils abgekürzt werden. Der Bezeichner „Finale“ kann ganz weggelassen werden, da im Fall keiner Angabe automatisch klar ist, dass es sich um ein Finale handelt. Für diese Konventionen diente der Zeitplan der Sparkassen Gala 2016 als Referenz. Um die Wörter auszutauschen, wurde der neue Datentyp Replacement erstellt, der das Schlüsselwort (key) und die passende Abkürzung (replacement) beinhaltet. Anschließend wurde das Array replaceKeys definiert, wo die entsprechenden Replacement-Objekte aufgeführt sind. Dies hat den Vorteil, dass man sofort erkennt was abgekürzt werden soll und die Daten einfach gehalten werden können. Des weiteren lassen sich schnell und einfach weitere abzukürzende Wörter in Zukunft hinzufügen. In der Methode wird die Ersetzung letztendlich durchgeführt. Als Parameterwert dient die zuvor jeweils festgestellte Disziplin eines StrucParagraph-Objekts. In der Methode

[21] http://www.w3schools.com/jsref/jsref_split.asp, abgerufen am 04.12.16

wird das Array replaceKeys durchlaufen und bei einer Übereinstimmung des keys mit discipline wird diese durch die Abkürzung (value) ersetzt.

Da die Disziplin erkannt und mögliche Abkürzungen eingeführt wurden, muss jetzt noch die Kategorie identifiziert. Die Kategorie setzt sich aus dem Geschlecht und der Altesklasse zusammen. Zuerst muss also festgestellt werden, ob es sich um einen Wettkampf für Frauen oder für Männer handelt. Um dies herauszufinden, wurde die erste Zeile mit Hilfe der split-Methode in ein Array aller vorhandenen Wörter zerlegt. Dies ermöglicht, dass man beim Durchlaufen das Array nach den Schlagwörtern „Frauen“, „weiblich“ und „weibliche“ durchsuchen kann. Man vergleicht dazu einfach den aktuellen Token (Leerraum wurde mit der trim-Methode entfernt) mit diesen Schlagwörtern. Diese Abfrage ermöglicht eine unkomplizierte Feststellung des Geschlechts. Wird eines dieser Schlagwörter gefunden, ist das Geschlecht weiblich („Frauen“). Ansonsten wird das Geschlecht auf männlich („Männer“) gesetzt. Nun muss die Altersklasse selbst noch identifiziert werden. Da in der ersten Zeile eines StrucParagraph-Objekts mehrere Kategorien enthalten sein können, muss die Variable, wo die Kategorien abgelegt werden sollen, ein Array sein. Für die Darstellung der Altersklassen wurden die offiziellen Bezeichner des SC DHfK – Abteilung Leichtathletik (Stand 2016) verwendet. Es gibt Haupt-, Jugend- und ältere Altersklassen.[22] Der erste Schritt ist, dass man das Vorkommen der Hauptklassen „Frauen“ und „Männer“ überprüft. Dazu durchsucht man das Array (brokenFLGender) wieder nach den verschiedenen Tokens. Wird eine Hauptklasse gefunden, wird diese sofort durch den push-Befehl in das Array category mitaufgenommen. Im Anschluss muss man noch das Vorkommen von den Jugendklassen überprüfen. Jugendklassen werden mit einem „M“ bzw. „W“ gefolgt von einer bzw. zwei Zahlen dargestellt. Da in der Ursprungsdatei aber diese Bezeichnung nicht zu finden ist, muss aus dem zuvor ermittelnden Geschlecht das Kurzzeichen erstellt werden. In der Ursprungsdatei werden die Jugendklassen immer durch ein „U“ gefolgt von einer bzw. Zahlen eingeleitet. D.h. man vergleicht den ersten Buchstaben des gerade betrachteten Tokens mit dem „U“. Ist der erste Buchstabe ein „U“, wird das Kurzzeichen und der Token als Kategorie zum Array hinzugefügt. Die letzte Möglichkeit ist, dass eine ältere Altersklasse noch in der ersten Zeile vorhanden ist. Diese werden mit einem „W“ bzw. einem „M“ eingeleitet. Es muss also wie bei den Jugendklassen nur der erste Buchstabe des Tokens auf Übereinstimmung geprüft werden und bei positivem Ergebnis die Kategorie zum Array hinzufügen.

Nachdem die Kategorie(n) erkannt wurde(n), muss aus dem Array category für jede Kategorie ein einzelnes Event (singleEvent) erstellt werden. Dabei sind die anderen Bestandteile des Events das Datum, der Veranstaltungsbeginn und die Disziplin. Diese werden wieder mit dem push-Befehl zum Array events hinzugefügt. Anschließend werden die events noch als JSON im sessionStorage gespeichert. Dieser Durchlauf wird für jeden Paragraphen wiederholt. Im letzten Schritt muss noch die automatische Weiterleitung zum HTML-Dokument GenerateTable integriert werden.

[22] https://www.leichtathletik-scdhfk.de/altersklassen/, abgerufen am 04.12.16

Kapitel 3

Schluss

Anhang

Anhang A

Erster Anhang

Anhang B

Zweiter Anhang

B.1 Anhang

B.2 Anhang

Literaturverzeichnis

Erklärung an Eides statt

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Regensburg, den 11. Februar 2016

Thomas Baumer, Benedikt Bruckner
Matrikelnummer 1721141, 1728475