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 Universal Character Set 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 bzw. tabellarischer Zeitplan in HTML-Form erstellt wird. 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. Darüber hinaus soll für den Zeitplan-Generator eine Desktop-App entwickelt werden.
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 also elimi-

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

niert, 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 beispielsweise 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 Projektdurchführung und Konzeption

2.1.1 Verwendete Hilfsmittel 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 wurde diskutiert wie das Projekt umgesetzt werden soll und welche Software bzw. Programmiersprache zur Generierung des Zeitplans verwendet werden sollen.
Da der Zeitplan im HTML-Format erstellt werden soll, benötigt man zum Erzeugen der Web-Dokumente und zum Verwenden von Web-Standards eine geeignete Programmiersprache und einen Editor. Zudem sind die Erstellung einer Desktop-App und eine ansprechende Oberflächengestaltung des Tools Ziele des Projektes, die erreicht werden sollen. Darüber hinaus wird für eine optimale Zusammenarbeit im Team eine Möglichkeit gesucht, die beispielsweise ein paralleles Arbeiten an den Problemen ermöglicht. Auf welche Art und Weise diese Ziele erreicht werden sollen, wird im Folgenden detaillierter erläutert.
Für die Wahl einer Skriptsprache wurde JavaScript verwendet. Vorteile dieser Löung sind, dass eine Vielzahl an Dokumentationen, Bibliotheken und Tutorials existieren. Beispielsweise kann bei der Programmierung auf die Bibliothek jQuery zurückgegriffen werden. Ein weiterer Vorteil von JavaScript ist, dass eine einfachere Einbindung (z.B. durch eine öffentlich zugängliche Website) möglich ist. Ein weiterer Grund für den Einsatz von JavaScript ist, dass die Studenten bereits erste Erfahrungen in JavaScript sammeln konnten. Die Verwendung von JavaScript ermöglicht die Erreichung aller gewünschten Ziele erreicht.
Als Editor zur Erstellung von HTML- bzw. JavaScript-Code wurde einerseits unter 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. Diese Editoren 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.
Für die Entwicklung einer Desktop-App wird die Software node.js in Verbindung mit dem Framework Electron eingesetzt. Dies bietet den großen Vorteil, dass für die Generierung des Zeitplans kein Browser benötigt wird. Deshalb ist es nicht notwendig im JavaScript-Code für die jeweiligen Browser umständliche Anpassungen zu definieren. Ein weiteres Argument für die Erstellung einer Desktop-App mit der Software node.js in Verbindung mit dem Framework Electron ist, dass nach dem Erstellen des eigentlichen Programms mit dem Editor nur noch kleine Anpassungen vorgenommen werden müssen, um ein fertiges Programm zu erhalten. Ein eigenständiges Programm erleichtert die Bedienbarkeit bzw. die Nutzerfreundlichkeit, da der Nutzer beispielsweise nicht die einzelnen HTML-Dokumente abspeichern muss. Des weiteren ermöglicht eine Desktop-App die Nutzung des Zeitplan-Generators, wenn kein Internetzugang vorhanden ist.
Abgesehen von der Funktionalität soll das Zeitplan-Tool auch eine ansprechende Oberfläche besitzen, da dies die Benutzerfreundlichkeit erhöht. Da der Zeitplan auch für große Leichtathletikveranstaltungen verwendet werden soll, ist die Oberflächengestaltung für die positive Wahrnehmung der Veranstaltung mitentscheidend. 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.[1] 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.[2]
Des Weiteren wurde die Verwendung einer Versionsverwaltungssoftware diskutiert. Eine Versionsverwaltung ermöglicht eine bessere Zusammenarbeit der Teammitglieder. Es wird beispielsweise das Wiederherstellen von alten Zuständen des Projekts und die Protokollierung ermöglicht. Es kann also jede Änderung an den Dateien mit Autor und Datum, also die ganze Versionsgeschichte, nachvollzogen werden. 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.[3] Deshalb wird für die Durchführung des Projektes eine Versionsverwaltungssoftware genutzt. Als Versionsverwaltungssoftware einigte man sich auf GitHub mit der Graphical user interface (GUI) des „GitHub Desktops“.
Nachdem alle für die Problemlösung benötigten Programme, Frameworks und Skriptspra-

[1] https://de.wikipedia.org/wiki/Bootstrap_(Framework),abgerufenam30.11.16
[2] http://www.w3schools.com/bootstrap/bootstrap_get_started.asp , abgerufen am 30.11.16
[3] Skript des Kurses Praxis des Programmierens SS16: Versionsverwaltung Folien 29ff.

chen festgelegt wurden, wird im Folgenden die Vorgehensweise zur Durchführung des Projektes beschrieben.

2.1.2 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 die Ursprungsdatei genauer analysiert. Es wurde festgestellt, dass die Ursprungsdatei sehr viele irrelevante Daten beinhaltet und man jeweils nur die zwei Zeilen im Absatz 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 korrekte bzw. dynamische Befüllung der Ergebnistabelle gelegt werden.

Ausgehend der gemeinsamen konzeptionellen Überlegungen im Team wurde zuerst der Rohtext spezifiziert, da dieser der Ausgangspunkt für die Entwicklung des Zeitplan-Generators ist. 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. Anschießend wurde das Problem den Zeitplan-Generator zu entwickeln in vier größere Teilprobleme bzw. Schritte unterteilt. 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. Der erste 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 und dadurch einzulesen. Der zweite Schritt ist, dass aus der Ursprungsdatei nur die relevanten Daten herausfiltert werden. Hier sollen praktisch nur noch alle relevanten Absätze betrachtet werden. Anschließend müssen diese Daten im dritten 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. 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. 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. Des weiteren werden für die Desktop-App noch die Dateien index.html, main.js und package.json benötigt. Die genaue Vorgehensweise in den einzelnen Schritten wird nun im Folgenden genauer erläutert.

2.1.3 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 Folgenden erläuterten Aufbau eines Paragraphen (-Tag), da nur die Paragraphen relevante Informationen für die Erstellung des Zeitplanes enthalten. Im listing ist beispielhaft der erste Paragraph in der zur Entwicklung verwendeten Ursprungsdatei aufgeführt.

```
<p class="ev1">
        <a name="21000001">
                100m (Vorprogramm), Frauen + U20 + U18 – Zeitläufe
        </a>
                <br>
        <span class="ev2">
                Datum: 05.06.2016  Beginn: 12:00
        </span>
</p>
```

Im Testfile ergibt sich folgender Klartext:

Zeile 1:	„100m (Vorprogramm), Frauen + U20 + U18 – Zeitläufe“
Zeile 2:	„Datum: 05.05.2016 Beginn: 12:00“

Daraus lässt sich eine verallgemeinerte Darstellung ableiten:

Zeile 1:	„Disziplin (Zusatzinformation), Kategorie(n) – Disziplin“
Zeile 2:	„Datum Beginn“

Diese zwei Zeilen sind DOM-Elemente der Form ... . Die relevanten Informationen werden mit Hilfe dieser Form identifiziert.
Die erste Zeile beinhaltet die Teilnehmerklasse(n) bzw. Kategorie(n) (hier: „Frauen + U20 + U18“) ungefähr in der Mitte der Zeile. Um die Teilnehmerklassen genau abzugrenzen, wird nach einleitenden bzw. beendenden Merkmalen der Teilnehmerklasse in der ersten Zeile gesucht. Die Teilnehmerklassen werden im Beispiel initiiert durch ein Komma

und beendet durch einen Bindestrich. Dieses Muster ist für alle Paragraphen zutreffend, weshalb die Kategorie auf diese Weise in der ersten Zeile identifiziert werden kann. Des weiteren ist es möglich, dass mehrere Kategorien in einem Paragraphen enthalten sind. Dabei sind die Kategorien durch ein „+“ -Zeichen voneinander getrennt. In der Kategorie ist zudem die Information über das Geschlecht der Teilnehmer vorhanden. Mit Hilfe der Zeichenfolgen „weiblich“, „weibliche“ oder „Frauen“ bzw. „männlich“, „männliche“ oder „Männer“ kann dies festgestellt werden, da diese Schlüsselwörter das Geschlecht in allen Paragraphen eindeutig kennzeichnen.
Ein weiterer Bestandteil der ersten Zeile ist die Disziplin. Die Disziplin setzt sich aus den ersten Zeichen bis zur öffnenden Klammer und den letzten Zeichen nach dem Bindestrich zusammen. Darüber hinaus kann der Inhalt der runden Klammern ebenfalls ein Teil der Disziplin sein. Ist eine der Zusatzinformationen „Hauptprogramm“, „Gala“, „Vorprogramm“ oder „Laufnacht“ der Inhalt der Klammern, ist die Zusatzinformation jedoch für die Bildung der Disziplin nicht relevant. Des weiteren gibt es die Möglichkeit, dass überhaupt keine runden Klammern mit Zusatzinformationen enthalten sind. Zusammenfassend ist die Zusatzinformation ein Teil der Disziplin, wenn eine Zusatzinformation im Paragraphen enthalten und diese nicht eines der ausgeschlossenen Wörter ist. Im Beispiel entsteht die Disziplin „100m Zeitläufe“. Da das Vorprogramm eine irrelevante Zusatzinformation ist, wird sie bei der Bildung der Disziplin vernachlässigt.
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.

Die für die Entwicklung verwendete Ursprungsdatei weist abgesehen von den enthaltenen Paragraphen 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. 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). Dieser weitere Aufbau der Ursprungsdatei ist jedoch für die Zeitplangenerierung nicht weiter relevant und kann vernachlässigt werden.
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.

2.2 Programmarchitektur und Ablauf

2.2.1 Allgemeine Festlegungen für die Web-Dokumente

2.2.1.1 Festlegung des Titels und Einbindung von Bootstrap

Wie in Punkt 2.1.2 beschrieben, werden zur Erstellung eines Zeitplan-Generators vier Web-Dokumente erzeugt. Diese Web-Dokumente haben alle einen HTML-Kopf (HEAD) und einen HTML-Körper (BODY), die jeweils auch durch die gleichnamigen Tags eingeleitet und beendet werden. Im HEAD wird immer der Titel des Web-Dokuments festgelegt. Darüber hinaus muss in den Dokumenten FileUploader und GenerateTable noch die Einbindung von Bootstrap im HEAD erfolgen, da diese Web-Dokumente ein Teil der Benutzeroberfläche sind. Dabei müssen für die Verwendung von Bootstrap immer drei Meta-Tags (<meta />) zu Beginn des HEADs deklariert werden. Dabei gibt das erste Meta-Tag den zu verwendenden Zeichensatz 8-Bit Universal Character Set 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.[4] 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.[5] Das zweite Meta-Tag wird ebenfalls standardmäßig definiert, um eine bestmögliche Darstellung im Internet Explorer, auch in den älteren Versionen, zu gewährleisten.[6] 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 mobilen 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.[7] Im Anschluss wird im HEAD die verwendete (minimierte) Bootstrap-CSS-Version definiert. Darüber hinaus werden in den Web-Dokumenten FileUploader und GenerateTable für die Verwendung von Bootstrap am Ende des BODYs noch die benötigten Skripte deklariert. Diese sind jQuery, das für Bootstrap JavaScript-Plugins benötigt wird, und das kompilierte JavaScript.[8]

2.2.1.2 Weiterleitung in den Web-Dokumenten

Im BODY wird standardmäßig die Weiterleitung zu den verschiedenen Dokumenten definiert. Während der Entwicklungsphase wurde jeweils ein Button implementiert, der

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

beim Anklicken zu den verschiedenen Web-Dokumenten weiterleitet. Des weiteren wurde standardmäßig ein displayArea festgelegt, das die verschiedenen Ergebnisse der Web-Dokumente anzeigt. So wurde ein einfaches Testen der Funktionalität und Debugging sicher gestellt. Die Weiterleitung zum jeweiligen Dokument erfolgt immer, wenn alle Ziele im vorherigen Schritt erreicht wurden. Die Web-Dokumente bauen also aufeinander auf und haben die Ergebnisse der vorherigen HTML-Files als Vorbedingung. Im Folgenden werden die genaue Abfolge und die Ziele in den einzelnen Schritte erläutert. Der FileUploader ist der Ausgangspunkt bei der Erstellung des Zeitplans. Hier wird das Inputfile, das in der spezifizierten Form vorliegen muss, eingelesen. Ist der Rohtext (rawText) im sessionStorage gespeichert, leitet das Programm zum nächsten Schritt bei der Generierung einer Tabelle, dem ParagraphHandler, weiter. Im ParagraphHandler 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. Sind diese Ziele erreicht, wird zum HTML-Dokument Cleaner weitergeleitet. Ziel im Cleaner ist es die Veranstaltungsdaten (Datum, Uhrzeit, Kategorie und Disziplin) als events zu extrahieren. Anschließend werden die events als JSON im sessionStorage gespeichert. Somit wurden die Veranstaltungen strukturiert ähnlich zu einer Datenbank abgelegt. Liegen die Events im sessionStorage vor, wird zum HTML-Dokument GenerateTable weitergeleitet. Im vierten zu erstellendem HTML-Dokument wird die Tabelle generiert.

Nach dem Abschluss der Programmierphase bzw. des Testens wurde der Submit-Button und gegebenenfalls das displayArea 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 (Auto-Submit) zu den jeweiligen Web-Dokumenten Zeit spart. Zudem ist es für den Nutzer irrelevant, dass er beispielsweise das eingelesene File mit den unstrukturierten Daten angezeigt bekommt. Das displayArea muss nur bei den relevanten Seiten, die zur Benutzeroberfläche gehören, angezeigt werden. Es wird also nur die Aufforderung ein File einzulesen mit dem dazugeörigen Button und der erstellte Zeitplan mit dem Speichern-Button angezeigt. Bei Bedarf kann das Auskommentieren natürlich wieder rückgängig gemacht werden, was das Debugging bzw. die Erweiterung des Tools vereinfachen würde.

2.2.1.3 Allgemeine Definierungen im JavaScript-Code

Im JavaScript-Code wurden die restlichen Funktionalitäten der Web-Dokumente umgesetzt. 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 zur Erstellung eines Zeitplan-Generators benötigt. Standardmäßig werden im

JavaScript-Code Methoden definiert, die das displayArea anzeigen und das Ergebnis des jeweiligen Web-Dokuments im sessionStorage abspeichern.

2.2.2 Funktionalität des File-Uploaders

2.2.2.1 Festlegung des Designs des File-Uploaders

Nachdem Bootstrap und der Titel im HEAD definiert wurden, wird im BODY die Gestaltung der Überschrift (Bootstrap Klasse "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-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. Außerdem passt diese Darstellung gut mit dem vorher definierten grauen Bereich zusammen. 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 den Input (type: file) ausgeführt wird, das unsichtbar bzw. versteckt ist. 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"). Dies ist eine saubere Lösung im Vergleich zu einschlägigen Vorschlägen aus dem Netz, wo der Fileinput mit einem Bild verdeckt wird. (vgl. http://www.quirksmode.org/dom/inputfile.html)

2.2.2.2 Einlesen und Abspeichern der Ursprungsdatei

Im JavaScript-Code wurde das Einlesen und Abspeichern der Ursprungsdatei umgesetzt. Dabei 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

[9] http://www.w3schools.com/bootstrap/bootstrap_jumbotron_header.asp, abgerufen am 02.12.16
[10] http://www.w3schools.com/Bootstrap/bootstrap_wells.asp, abgerufen am 02.12.16

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 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 Methode onload bzw. das Attribut result der File Reader-API zunutze. Onload erkennt, wann das File komplett eingelesen wurde und result gibt den Inhalt des Files zurück.[11] 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. Es ist also eine spätere Weiterverwendung der Zwischenergebnisse gewährleistet. 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.

2.2.3 ParagraphHandler: Herausfiltern der Paragraphen aus dem Rohtext

2.2.3.1 Einbindung von jQuery

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 Anschluss 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

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

zu bearbeiten. jQuery ermöglicht also DOM-Abfragen mit einer einfachen Syntax und bietet zudem Methoden, die wichtige Aufgaben mit sehr wenig Code erledigen.[12] Bei der Einbindung von jQuery in den Paragraph Handler gibt es drei verschiedene Möglichkeiten. Eine Möglichkeit ist jQuery über Google zu laden. Dies hat den großen Vorteil, dass jQuery nicht extra geladen werden muss, wenn vorher Websites jQuery ebenfalls über die Google-API eingebunden haben. Dies macht sich natürlich auch in der Performance bemerkbar. Muss jQuery trotzdem geladen werden, ist dies durch den Google-Server schnell möglich. Eine weitere Lösung ist, dass man die aktuellste Version von jQuery.com nutzt. Hier kann einfach der Link der aktuellsten jQuery-Version als Quelle angegeben werde. Dies hat den Vorteil, dass man automatisch immer die neueste Version verwendet. Die dritte Möglichkeit ist die Einbindung von jQuery über den lokalen Pfad, wo die Version nach dem Herunterladen abgespeichert wurde. Dieser Weg ist zwar nicht sehr komfortabel, aber der große Vorteil gegenüber der anderen Möglichkeiten ist, dass man keine Internetverbindung benötigt. Da die Tabelle auch bei lokaler Einbindung schnell generiert werden kann und die Verwendung der aktuellsten Version keine Rolle spielt, hat man sich für die lokale Einbindung entschieden.[13] Ist der Nutzer beispielsweise gerade auf einer Leichtathletikveranstaltung und will sich über den Ablauf der Wettkämpfe informieren, muss er für die Einbindung von jQuery nicht online sein. Darüber hinaus muss jQuery als Skript für Electron und Node.js angegeben werden.

2.2.3.2 Identifizierung und Speichern der relevanten Paragraphen

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.[14] 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.[15] 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,

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

die den Inhalt ($html) am Beginn der selektierten Elemente (pre) einfügt.[16] Da nun der Textinhalt der Paragraphen im displayArea angezeigt wird, kann der Inhalt nun mit der saveParagraphText(paragraphText)-Methode im sessionStorage abgespeichert werden.

2.2.3.3 Vorbereitungen für ein strukturiertes Abspeichern der Paragraphen

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, 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 als Konstante 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

[16] http://www.w3schools.com/jquery/html_prepend.asp, abgerufen am 27.11.16

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. Die Methoden in Verbindung mit dem regulären Ausdruck wurden aber verwendet, da die Paragraphen in der ersten und zweiten Zeile genug Regelmäßigkeit aufweisen, um diese Lösung zu benutzen. Die Methoden ermöglichen ein schnelles und einfaches Erhalten der gewünschten Positionen im Paragraphen. Mit Hilfe der Methode search wird nun nach der im regulären Ausdruck definierten Form im Text gesucht und die Position der Übereinstimmung zurückgegeben.[17] Da die erste Stelle der Uhrzeit als Position 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 kopiert. Dabei braucht die slice-Methode als Parameterwerte einen Start- und Endpunkt, wobei der Endpunkt nicht mehr Teil des Ergebnisstrings ist.[18]

2.2.3.4 Unterteilung der Paragraphen in erste und zweite Zeile

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 Para-

[17] http://www.w3schools.com/jsref/jsref_search.asp, abgerufen am 27.11.16
[18] http://www.w3schools.com/jsref/jsref_slice_array.asp, abgerufen am 27.11.16

graph 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.[19] Zudem ermöglicht die Methode stringify(value) ein simples Parsen zu einem String. Das Array wird also als String in der Variable arrayJSON abgespeichert.

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

2.2.4.1 Vorbereitungen für die Identifizierung der Daten

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.
Dies wird im JavaScript-Code umgesetzt, der im Folgenden genauer erläutert wird. Um mit den Paragraphen zu arbeiten, muss zunächst das mit JSON im sessionStorage abgespeicherte Array (als String) 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 mit JSON 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 oder mehrere Events abgespeichert. Dies hat den Vorteil, dass die Daten korrekt 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

[19] JavaScript & jQuery - Interaktive Websites entwickeln, von Jon Duckett S.374

Paragraphen auf einmal umgesetzt werden kann. Die einzelnen Teilprobleme nicht auszulagern erscheint zwar unübersichtlich, aber diese Methodik erlaubt gleichzeitig, dass man im Verlauf die Daten in der Kopie(z.B. von der ersten Zeile) einfach verändern kann. 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.

2.2.4.2 Identifizierung des Datums und der Uhrzeit

Die erste und zweite Zeile lassen sich prinzipiell unabhängig voneinander bearbeiten. 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 Datums dient. D.h. Sonderfälle wie der Februar, der keine 3 an erster Stelle bei der Angabe des Tages haben kann, werden nicht auf Korrektheit 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 Kapitel 2.2.3.3 genauer erläutert wurde. Mit der Kombination aus search- und slice-Methode erhält man wieder das gewünschte Ergebnis (hier: die Uhrzeit) geliefert.

2.2.4.3 Identifizierung der Disziplin

Jetzt müssen die Daten der ersten Zeile noch identifiziert werden. Zuerst wird die Disziplin identifiziert. Da in der ersten Zeile in Klammern relevante oder irrelevante Zusatzinformationen für die Disziplin 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 cutOutOfBrackets abgespeichert. Um diesen Fall im Code abzubilden, wurde zuerst eine if-Bedingung definiert. Diese wird nur durchlaufen, wenn Klammern enthalten sind. Sind Klammern vorhanden, wird mit Hilfe der split-Methode die Zeile in drei Teile zerlegt.[20] 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 irrelevante Zusatzinformation enthalten ist. Dazu muss das Array cutOutOfBrackets durchlaufen werden. Ist eine irrelevante Zusatzinformation vorhanden, wird diese gelöscht. Falls eine relevante Zusatzinformation gefunden wird, wird diese zwischengespeichert. 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.
Die Disziplin setzt sich aus mindestens einem Teil der ersten Zeile zusammen. Dabei ist der Teil vom Beginn der Zeile bis zum Komma (Klammern und Inhalt der Klammern sind nicht mehr vorhanden) immer ein Bestandteil der Disziplin. Der zweite Teil setzt sich gegebenenfalls aus dem Ende der ersten Zeile, dem Teil nach dem Bindestrich, zusammen. Darüber hinaus kann der Inhalt der Klammern, wenn dieser als relevant eingestuft wurde, 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. Existiert der Klammerninhalt nicht oder wurde er als irrelevant eingestuft, kann die Disziplin aus dem ersten und gegebenenfalls

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

dritten Teil bereits zusammengesetzt werden. Ist der Klammerninhalt relevant, wird dieser 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. Prinzipiell ist hier eine ähnliche Lösung wie bei cutOutofBrackets möglich bzw. die Funktionalität der Methode wäre als Replacement-Lösung abbildbar. Da aber hier nicht einmal ein Schlagwort gefunden werden muss, sondern nur Wörter verglichen und gegebenenfalls gelöscht bzw. ausgetauscht werden müssen, wird ein Array verwendet. 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 wird das Array replaceKeys durchlaufen und bei einer Übereinstimmung des keys mit discipline wird diese durch die Abkürzung (value) ersetzt.

2.2.4.4 Identifizierung der Kategorie

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. Das Geschlecht wird benötigt, da ein Paragraph immer den Wettkampf für ein Geschlecht plus die verschiedenen Kategorien darstellt. 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 Al-

tersklassen wurden die offiziellen Bezeichner des SC DHfK – Abteilung Leichtathletik (Stand 2016) verwendet. Es gibt Haupt-, Jugend- und ältere Altersklassen.[21] 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 einem „U“ und 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 (W oder M) auf Übereinstimmung geprüft werden. Bei einem positivem Ergebnis wird die Kategorie zum Array hinzugefügt.

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. Somit wurden die Veranstaltungen strukturiert ähnlich zu einer Datenbank abgelegt. Dieser Durchlauf wird für jeden Paragraphen wiederholt.

2.2.5 GenerateTable: Dynamische Erstellung des Zeitplans

2.2.5.1 Gestaltung des Speichern-Buttons

Da die Tabelle dem Nutzer angezeigt werden soll und zusätzlich ein Speichern-Button integriert wird, muss für eine ansprechende und konsistente Darstellung wieder Bootstrap im Web-Dokument integriert werden. Im BODY wird nun der Speichern-Button, der dem Nutzer das lokale Speichern des Zeitplans ermöglicht, gestaltet. Diese Funktion ist für den Benutzer sehr sinnvoll, da er bei mehrmaligen Betrachten nicht immer die Website bzw. das Tool aufrufen muss. Der Speichern-Button soll für eine einheitliche Darstellung das gleiche Design wie der „Datei auswählen“-Button beim FileUploader haben. Der Button wird also wieder durch einen grauen Bereich vom restlichem Dokument abgehoben (Bootstrap Well-Klasse) und als Button-Style wurde wieder auf den Info-Button zurückgegriffen (vgl. 2.2.2.1 Festlegung des Designs des File-Uploaders).
Wurde die Tabelle generiert, hat der Nutzer noch die Möglichkeit diese durch den Klick

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

auf den Button lokal abzuspeichern. Dabei wurden drei verschiedene Möglichkeiten für die Umsetzung des Speicherns diskutiert. Eine Möglichkeit ist, dass die Tabelle (<table>...</table>), die im sessionStorage gespeichert wurde, heruntergeladen wird. Eine weitere Lösung ist, dass man das komplette HTML zum lokalen Speichern anbietet. Hier ist es dem Nutzer möglich das HTML-Dokument erneut runterzuladen. Die dritte Möglichkeit ist, dass man das gesamte HTML-Dokument mit dem ausgeblendeten Button zum Abspeichern freigibt. Dies hat den Vorteil, dass die lokale Kopie nur den eigentlichen Inhalt (die Tabelle) enthält. Zudem bleibt durch das Ausblenden die Funktionalität des Buttons erhalten. Ein mehrfaches Abspeichern ist bei dieser Lösung ebenfalls möglich. Aufgrund dieser Vorteile wird als Export das gesamte HTML-Dokument mit dem ausgeblendeten Button gewählt. Die Funktionaliät wird in der Methode createLocalCopy() umgesetzt. Da man sich für die dritte Möglichkeit entschieden hat, wird zuerst der Button ausgeblendet. Im Anschluss wird der komplette Seiteninhalt in der Variable text gespeichert. Darüber hinaus wurde der Name der lokalen Kopie (Zeitplan.html) und der Typ des Dokuments (HTML) festgelegt. Anschließend wird ein neues File erstellt und es wird ermöglicht dieses lokal abzuspeichern.

2.2.5.2 Sortierung der Events

Im JavaScript-Code werden im ersten Schritt die als JSON im sessionStorage gespeicherten Events ausgelesen und in Event-Objekte geparst. Im Anschluss müssen die Event-Objekte sortiert werden. Da für jeden Wettkampftag eine eigene Tabelle erzeugt wird, müssen die Events zuerst nach dem Kriterium Datum sortiert werden. Die Sortierung zweiter Ordnung erfolgt nach der Zeit, da alle Wettkämpfe an einem Tag aufsteigend nach dem Beginn sortiert werden sollen. Darüber hinaus sollen die Events nach den Altersklassen geordnet im Zeitplan angezeigt werden. Es muss also noch nach der Kategorie sortiert werden. Die Sortierung nach diesen drei Kriterien wird in der ausgelagerten Methode sortEventsDateAndTimeAndCategoryOrder(events) vorgenommen.

Im ersten Schritt muss eine Möglichkeit zum Vergleichen der drei Kriterien mit den verschiedenen Werten eines Events gefunden werden. Durch die Umwandlung des Datums, der Uhrzeit bzw. der Kategorie zu einem Integer, kann man die entstehenden Werte leicht miteinander vergleichen. Das Ziel ist es also aus den drei Kriterien einen Vergleichswert (Integer) der folgenden Form zu generieren:

	Jahr	Monat	Tag	Stunden	Minuten	Kategorie
Beispiel	2016	06	05	12	15	116

Anmerkung: Kategorie soll durch einen dreistelligen Integer dargestellt werden

Zuerst wird mit Hilfe der split-Methode das Datum und die Uhrzeit in ein Array aus Substrings aufgeteilt. Dabei werden für die Unterteilung der Punkt bzw. der Doppelpunkt verwendet. Nun ist es möglich das Datum und die Uhrzeit absteigend nach dem Jahr, dem Monat, dem Tag, den Stunden und den Minuten anzuordnen. Nach der Kategorie wird in

der Methode getSortKeyTableHead(category) sortiert. Diese Funktion wurde ausgelagert, da sie mehrmals benötigt wird. Die Kategorien sollen folgendermaßen geordnet sein: weibliche Jugendklassen, Frauen, ältere Frauenklassen, männliche Jugendklassen, Männer und ältere Männerklassen.
Der Algorithmus soll also für die jeweilige Klasse einen dreistelligen Integer zurückgeben, der die Altersklasse in das gewünschte Muster einordnet. Dabei werden für die jeweiligen Klassen bestimmte Basiswerte festgelegt. Für die Hauptklassen „Frauen“ und „Männer“ wird 200 bzw. 600 zurückgegeben. Diese Zahlen wurden so festgelegt, damit für die Jugend- und älteren Klassen genug freie Werte zum Einordnen bleiben. Die weiblichen bzw. männlichen Jugendklassen wurden initial mit dem Wert 100 bzw. 500 belegt. Dabei wird die jeweilige Zahl der Altersklasse zu 100 bzw. 500 addiert. Nach dem gleichen Prinzip wird auch bei den älteren Klassen ein Integer erstellt, wobei die Werte für die weiblichen Klassen bei 300 und für die männlichen Klassen bei 700 beginnen. Kann der Token keiner Kategorie zugeordnet werden und ist der Token nicht „Zeit“, wird der Wert an der rechten Seite der Tabelle hinzugefügt. Es wird der Integer 999 zurückgegeben. Nun ist ein Integer entstanden, der ein Vergleichen der Events ermöglicht. Durch einfaches Subtrahieren von einem Integer erhält man einen Indikator zum Sortieren/Tauschen des Events. Die Methode sortEventsDateAndTimeAndCategoryOrder(events) gibt die sortierten Events zurück.

2.2.5.3 Gruppierung nach tagesgleichen Events

Nach dem Sortieren werden die Events auf die verschiedenen Tage aufgeteilt. Dies erfolgt in der Methode splitEvents(events). Es wird ein zweidimensionales Array initialisiert. In der ersten Dimension ist das Array nach dem Tag aufgeteilt und in der zweiten Dimension beinhaltet das Array ein weiteres Array aus den tagesgleichen Events. Die Variable indexReturnArray ist der Index der ersten Dimension, der angibt an welcher Stelle der aktuelle Tag ist. Die Variable currentDate soll das Datum der aktuell betrachteten Gruppierung (das zu erstellende tagesgleiche Event-Array) definieren. Nun wird für jedes Event aus dem übergebenen events-Array überprüft, ob es den selben Tag besitzt wie das letzte Element, das betrachtet wurde. Dabei wird currentDate mit events[i].date verglichen. Wenn das Datum des zu erstellenden tagesgleichen Arrays (currentDate) mit dem des aktuell betrachtetem Event (events[i].date) übereinstimmt, kann das aktuell betrachtete Event ins returnArray in die zweite Dimension (an der Stelle indexReturnArray), wo die Events sind, gepusht werden. Haben die Variablen einen unterschiedlichen Tag, wird die zweite Dimension um eine neue „Zeile“ erweitert. Der Zeiger indexReturnArray wird um eins erhöht und an dieser Stelle (neue Zeile bzw. den nächsten index im returnArray (ersteDimension)) wird ein neues Array initialisiert. Hier wird das aktuell betrachtete Event abgelegt und currentDate wird auf das neue Datum aktualisiert, sodass beim nächsten Vergleichen mit events[i].date wieder festgestellt werden kann, ob eine weitere Gruppierung benötigt wird.

2.2.5.4 Generierung der Tabelle

Im Folgenden wird die Tabelle und die Überschrift für jeden Tag (eventsSplitDays) erstellt, wobei für jeden Durchlauf der for-Schleife eine Überschrift und eine Tabelle erstellt wird. Zuerst wird dem Seiteninhalt (pageContent) die Überschrift „Tagesprogramm für den“ und das aktuelle Datum der zugehörigen Events hinzugefügt. (div-Tag) Dabei soll auch wieder der graue Bereich die Überschrift umgeben. (Bootstrap Well-Klasse) Anschließend muss die Tabelle selbst noch generiert werden, was in der Methode generateTable(events) umgesetzt wird. Als Parameterwert bzw. Input müssen die sortierten und tagesgleichen Events vorliegen.
Das Ziel von generateTable(events) ist es als Output eine Tabelle (ohne Überschrift) mit dynamischen Inhalt und Tabellenkopf zu erzeugen. Die Zeilen und Spalten sind zusammen der Schlüssel für den Inhalt, wobei auch mehrere Events denselben Schlüssel haben können. Es können also auch zwei Events in einer Zelle stehen. Beim Aufruf generateTable(events) wird zunächst die Tabelle initialisiert. Die Tabelle wird wieder mit Bootstrap gestaltet. Dabei soll die Tabelle die Form der Bootstrap Hover Rows-Tabelle haben. Diese Klasse fügt beim Zeigen auf die jeweilige Reihe der Tabelle einen Hover-Effekt (graue Hervorhebung) hinzu.[22] Dieser Effekt unterstützt einerseits das Design und hilft andererseits dem Nutzer beim Finden der gewünschten Tabelleneinträge.

Anschließend wird der Tabellenkopf generiert, den die Methode getTableHead(events) liefert. Als erstes wird ein Array (tableHead) definiert, das als Rückgabewert des Tabellenkopfs dienen soll. Zum Array wird als erstes „Zeit“ hinzugefügt, da diese Spalte bei jedem Tabellenkopf gleich sein soll. Anschließend sollen alle im Event-Array vorkommenden Kategorien ein Teil des Tabellenkopfs sein, wobei jede Kategorie jeweils nur einmal im Tabellenkopf enthalten sein soll. Um die Kategorien dynamisch zum Array hinzuzufügen, wird für jedes Event geprüft ob die Kategorie schon hinzugefügt wurde. Wenn eine neue Kategorie gefunden wird, wird diese dem Array hinzugefügt. Wenn alle Daten für den Tabellenkopf herausgefunden wurden, müssen diese noch sortiert werden. Für die richtige Sortierung nutzt man wieder die Methode getSortKeyTableHead(token), die für die Vergleichbarkeit der Token einen Integer liefert. Anschließend kann durch das Ergebnis der Subtraktion zweier Integer (sortKeyA, sortKeyB) herausgefunden werden, ob die Werte getauscht werden müssen. Da bei der Akquirierung der Daten darauf geachtet wurde, dass die Werte einzigartig sind, können sie nicht übereinstimmen. Wurden alle Werte richtig sortiert, wird das Array „tableHead“ zurückgegeben. Nachdem die Daten für den Tabellenkopf ermittelt wurden, muss aus diesen Daten der HTML-Tabellenkopf erzeugt werden. Die Methode generateTableHead(tableHead) wandelt das Array tableHead zu einem HTML-String um, der der Tabelle hinzugefügt werden kann. Es wird festgelegt, dass der Tabellenkopf genau eine Zeile sein soll (tr-Tag). Anschließend wird das Array tableHead durchlaufen und es können die Tabellenkopfeinträge in den th-Tags eingefügt werden. Der Anfang und das Ende einer Zelle wird durch das th-Tag definiert. Wurden

[22] http://www.w3schools.com/bootstrap/bootstrap_tables.asp, abgerufen am 08.12.16

alle Einträge hinzugefügt, wird die für den Datensatz minimale und dynamische Tabellenkopfzeile zurückgegeben.

Neben dem Tabellenkopf muss aus den Events noch der Tabellenkörper generiert werden. Zuerst wurden die Objekte „Row(time, cells)“ und „Cell(category, content)“ definiert. Diese Objekte sollen eine Zeile/Reihe (row) bzw. eine Zelle repräsentieren und werden für die Erstellung des Tabellenkörpers benötigt. Die Erstellung wird in der Methode generateTableBody(tableHead, events) umgesetzt. Zuerst sollen von allen Events (bereits nach dem Tag gruppiert) diejenigen Events herausgefunden werden, die zur selben Uhrzeit an dem jeweiligen Tag stattfinden. Diese Events müssen in die gleiche Zeile geschrieben werden. Die Methode getGroupedEvents(events) arbeitet nach dem gleichen Prinzip wie die Methode splitEvents(events). Der Unterschied ist, dass jetzt die Events mit dem gleichen Beginn und nicht mit dem gleichen Datum gruppiert werden. Es entsteht also ein Array, indem die verschiedenen Positionen die zeitgleichen Events wiederspiegeln. Im Anschluss daran werden die zeitgleichen Events (groupedEvents) zu einer „TableRow“ (row-Objekt) geparst. Es soll ein Datenobjekt generiert werden, der einer Tabellenzeile ähnelt. Die Umwandlung der groupedEvents erfolgt in der Methode parseToRows(groupedEvents). Für alle zeitgleichen Events wird also eine Zeile erstellt, wobei ein Event in einer Zelle (Cell) steht. In einer Zelle wird die Kategorie und die Disziplin abgespeichert. Diese Zellen werden in einem Array (currentCells) zwischengespeichert. currentCells beinhaltet für eine Gruppierung von groupedEvents alle Kombinationen aus Kategorie und Disziplin, die aus den Events herauszulesen sind. Da dadurch die Kategorien redundant sein können, muss das Array gekürzt werden. Kürzen bedeutet in diesem Fall, den Zelleninhalt der redundanten Kategorien zu mergen. Die Methode shortCells(cells) liefert die gekürzten Zellen als Array zurück. Es werden alle Zellen dem Array hinzugefügt, abgesehen von den Zellen, die sich nur in der Disziplin unterscheiden. Als Primärschlüssel für dieses Zellen-Array wird also die Kategorie verwendet. Für diese Zellen muss der jeweilige Inhalt zusammengefügt werden, wobei zwischen den Inhalten ein Zeilenumbruch erfolgen soll. Nachdem die gekürzten Zellen vorliegen, kann eine Reihe aus dem Zellen-Array und der Uhrzeit erstellt werden. Dieser Vorgang wird für alle Tabellenzeilen wiederholt, wodurch das Array rows entsteht. Zurück in der Methode generateTableBody(tableHead, events) wird nun als String der Tabellenkörper aus den rows erstellt. Dazu wird dem Ergebnis HTML-String immer die nächste Zeile aus rows, nachdem sie in getTableRow(row[i], tableHead) in das HTML-Format gebracht wurde hinten angefügt. Die Tabellenzeilen müssen als HTML-String dargestellt werden, was in der Methode getTableRow(row, tableHead) umgesetzt wird. Eine Zeile wird durch das tr-Tag initialisiert. Als erster Eintrag einer Reihe soll die Uhrzeit der gerade betrachteten Zeile stehen (td-Tag). Im Folgenden sollen die restlichen Zellen von links nach rechts anhand der aktuellen Kategorie erzeugt und gegebenenfalls befüllt werden. Für jede Spalte wird in der Reihe eine Zelle erstellt, egal ob diese Zelle als Inhalt eine Disziplin hat oder nicht (td-Tag). Mit der Methode getTableCellContent(cells, currentCategory)

wird der Inhalt der gerade betrachteten Zelle geprüft, ob die betrachtete Zelle einen Inhalt hat. Wenn die Zeile eine Disziplin als Inhalt hat, wird diese zurückgegeben und zwischen den td-Tags eingefügt. Enthält die Zelle keinen Inhalt, wird ein leerer String zurückgegeben und zwischen den td-Tags eingefügt. Nach dem Aufruf dieser Methode, kann die Zelle durch das schließende td-Tag abgeschlossen werden. Wurde die komplette Reihe (row-Objekt) abgearbeitet, wird die Zeile durch das schließende tr-Tag beendet.

Nachdem der Tabellenkopf und der Tabellenkörper generiert wurden, kann in der generateTable(events)-Methode die Tabelle beendet (</table>) und die komplette Tabelle zurückgegeben werden. Der komplette Seiteninhalt wurde erstellt, wenn für alle tagesgleichen Events eine Tabelle erzeugt wurde. Ist das der Fall, wird die Tabelle im sessionStorage gespeichert und anschließend im displayArea angezeigt.

2.2.6 Erstellung einer Desktop-App für den Zeitplan-Generator

Um mit Electron eine Desktop-App zu erstellen, werden die Dateien index.html, main.js und package.json benötigt. Die index.html-Datei beinhaltet den Content der Seite, main.js liefert die JavaScript-Logik und package.json hinterlegt alle wichtigen Eckdaten zu der App.[23] Für die Erstellung der Desktop-App wird das Quick-Start-Electron-Tutorial als Hilfestellung verwendet.[24] Im Folgenden werden die drei benötigten Dateien näher erläutert.

In dem Dokument index.html wird der Inhalt der Website, der angezeigt werden soll, festgelegt. Im HEAD wird neben den standardmäßigen Deklarierungen des zu verwendenden Zeichensatzes UTF-8 und des Titels das Icon der App festgelegt. Ein eigenes Icon ist bei vielen Desktop-Apps ein Standard, weshalb für den Zeitplan-Generator auch ein eigenes Icon gestaltet wird. Zudem erscheint das Tool mit einem Icon benutzerfreundlicher. Das Icon wird mit der Software Iconion erstellt, die eine Vielzahl an Symbolen, Styles und Einstellungsmöglichkeiten zur Icon-Gestaltung frei zur Verfügung stellt.[25] Das Icon wird ähnlich wie Teile der Oberfläche der Web-Dokumente FileUploader und GenerateTable wieder hellblau gestaltet. Zudem ist ein Dokumentensymbol in weißer Farbe Bestandteil des Icons, das den Zeitplan widerspiegeln soll. (vgl. Abbildung 2.1) Da das Web-Dokument FileUploader.html der Ausgangspunkt des Zeitplan-Generators ist, wird dieses im BODY als das Ausgangsfile für die Desktop-App definiert. Im JavaScript-Code wird ein Auto-Submit deklariert, der automatisch zu den Web-Dokumenten weiterleitet.

In der Datei package.json werden wichtige Informationen hinterlegt. Es werden standardmäßig der Name, der Produktname, die Version, das main-Feld, die Abhängigkeiten (dependencies) und drei Skripte definiert. Das main-Feld gibt das Script, welches den Hauptprozess (main.js) auslöst, an. Unter Abhängigkeiten versteht man die Angabe

[23] http://t3n.de/news/electron-github-nodejs-chromium-656328/, abgerufen am 20.12.16
[24] http://electron.atom.io/docs/tutorial/quick-start/, abgerufen am 20.12.16
[25] http://iconion.com/de/, abgerufen am 20.12.16

Abbildung 2.1: Icon der Desktop-App

der benötigten Versionen von jQuery, electron und des electron-packagers. Die Skripte werden für die Betriebssysteme Mac OS, Windows und Linux definiert. So wird die Ausführbarkeit der Desktop-App auf diesen Betriebssystemen gewährleistet. Dies ist ein großer Vorteil, da nicht für jede Umgebung eine eigene App programmiert bzw. viele Anpassungen vorgenommen werden müssen.

Die dritte Datei main.js liefert die JavsScript-Logik für die Desktop-App. Dies beinhaltet das Erzeugen von Fenstern und das Handeln von System-Events. Zuerst werden eine Referenz für ein Fenster-Objekt und das Icon definiert. Die Größe des Browserfensters, wo die HTML-Dokumente ausgeführt wird festgelegt. Zudem wird der Content der Desktop-App, index.html, geladen. Anschließend wird definiert, wann die Browserfenster erzeugt bzw. beendet werden. Das Browserfenster wird erzeugt, sobald Electron die Initialisierung abgschlossen hat. Die App wird beendet, wenn alle Browserfenster geschlossen wurden. Hier wird noch eine Ausnahme für Mac OS definiert, da hier die Programme standardmäßig erst durch das Kommande Cmd+Q geschlossen werden. Wird in der App-Version für Mac OS wieder auf das Icon geklickt, wenn die App nicht geschlossen ist, wird das Fenster der App wieder erzeugt.

Kapitel 3

Schluss

Anhang

Anhang A

Erster Anhang

Anhang B

Zweiter Anhang

B.1 Anhang

B.2 Anhang

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