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

DOM	Document Object Model
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

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 werden die Daten für die Erstellung eines Zeitplans der Veranstaltungen der LG Telis Finanz aus einer Teilnehmerliste bzw. parallel in einem manuell gepflegten Zeitplandokument manuell ins HTML-Format übertragen. Da das Dokument mit den relevanten Daten sehr unstrukturiert ist und viele überflüssige Informationen enthält, ist der Zeitplan natürlich sehr aufwändig zu erstellen. Zudem entstehen durch die redundante Datenhaltung manchmal inkonsistente Zeitplandaten in der Teilnehmerliste und im manuell gepflegten und ins HTML-Format übertragenen Dokument.

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 Hypertext Markup Language (HTML)-Ursprungsdatei 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.

Ein sehr motivierender 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 eliminiert, da beispielsweise keine Tippfehler oder falsche bzw. fehlende Eintragun-

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

gen 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. Diese Punkte werden natürlich an den Termin der Veranstaltung ausgerichtet und im schlimmsten Fall hätte das 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 Eingangsfile 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 Schema 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 die Entwicklung eines Zeitplan-Generators für Leichtathletikveranstaltungen rechtfertigen. 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 (beide 5. Semester Wirtschaftsinformatik) 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 Programmiersprache 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 zurückgegriffen werden. Zudem wird ein klares und übersichtliches Programmieren sowohl bei JavaScript, als auch bei Python ermöglicht. Da Thomas Baumer und Benedikt Bruckner aber bereits im Kurs „Internettechnologien und Network-Computing“ erste Erfahrungen in JavaScript sammeln konnten, entschied man sich für JavaScript und sparte sich so eine größere Einarbeitungszeit in eine neue Programmiersprache. 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 den „GitHub Desktop“.

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, jQuery 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. Der Vorteil hier 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.

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.[2] 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 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.

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

[1] Skript des Kurses Praxis des Programmierens: Versionsverwaltung Folien 29ff.
[2] https://de.wikipedia.org/wiki/LaTeX, abgerufen am 21.11.16

wurde natürlich das Ursprungsfile genauer betrachtet. Es wurde festgestellt, dass das Eingangsfile 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 die Aufschlüsselung 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 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 Eingangsfiles 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 des Ursprungsfiles. Um mit dem Rohtext zu arbeiten muss dieser natürlich zuerst vom Programm identifiziert und eingelesen werden. Hier ist es die Aufgabe des Nutzers das gewünschte File, das die zuvor spezifizierte Form aufweist, auszuwählen. Der dritte Schritt ist, dass man aus dem Ausgangsfile nur die relevanten Daten herausfiltert. Hier sollen praktisch nur noch alle relevanten Absätze angezeigt 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 werden. Im letzten Schritt wird schließlich der gewünschte Zeitplan aus den zuvor identifizierten Informationen 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 Eingangsfile vier HTML-Files 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. einfacheres 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 Hilfe des Document Object Model (DOM). Im Folgenden wird nun der Inhalt und die Struktur des Eingangsfiles näher spezifiziert. Grundsätzlich kann der Rohtext beliebigen HTML-Code beinhalten mit der Ausnahme des im später erläuterten Aufbau eines Paragraphen. Das für die Entwicklung verwendete Ursprungsfile weist beispielsweise folgenden Aufbau auf. In den ersten zwei Zeilen des Files steht die Überschrift („Laufnacht und Sparkassen Gala 2016“) und der Ort bzw. das Datum („Regensburg, von 04.06.2016-05.06.2016“) geschrieben. Daraufhin folgen verschiedene Verweise, wie die Altersklassen (z.B. „Männer“ und „weibliche Jugend U20“) und alle Wettkampfbezeichnungen (z.B. „100m (Vorprogramm) - Zeitläufe“). Im Anschluss kommen die relevanten Zeilen für die Erstellung des Zeitplanes. Im Testfile beispielsweise: - 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 im verwendeten Ausgangsfile immer eine Tabelle mit der Startnummer, dem Namen, dem Jahrgang, der Nationalität, dem Verein, der Saisonbestleistung und der persönlichen Bestleistung, sowie den verschiedenen

Einträgen der Wettkampfteilnehmer. 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 erstellt. 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. Im HEAD werden lediglich Informationen des Web-Dokuments wie in diesem Falle der Titel und der zu verwendende Zeichensatz 8-Bit UCS Transformation Format (UTF-8) als Metainformation definiert. Die Verwendung von UTF-8 als Zeichensatz hat den Vorteil, dass alle Zeichen (auch Fremdwörter und Sonderzeichen) beliebig verwendet werden können.[3] 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 rechtfertigt.[4] Im BODY wurde der Nutzerhinweis „Read in the rawTable!“ und der Inputtyp festgelegt. Dabei wurde bestimmt, dass der Input vom Typ eines Files sein soll. Dies wurde mit dem div-Element definiert, was ermöglicht, dass die Inhalte des Files sauber in einen Bereich dargestellt werden. Außerdem 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. Die Benutzereingaben werden beim Absenden des Formulars an die im action-Attribut spezifizierte Adresse (ParagraphHandler.html) geschickt. Dabei werden die Daten erst versendet, wenn diese mit dem Formular-Element Submit-Button bestätigt wurden. (Nach Debugging/Programmierung auskommentieren des Buttons und automatische Weiterleitung). Zudem wurde im HTML-Körper noch die Bildschirmausgabe (displayArea) deklariert, die eventuelle Ausgaben anzeigt.

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 wer-

[3] https://wiki.selfhtml.org/wiki/Zeichenkodierung, aberufen am 23.11.16

[4] https://de.wikipedia.org/wiki/UTF-8, abgerufen am 23.11.16

den 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 Fileauswahl 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 die Bedingungen 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. 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.[5] 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. 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.

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

2.5 Paragraph Handler: Herausfiltern der relevanten Informationen aus dem Rohtext

In diesem Schritt ist es das Ziel aus dem im sessionStorage gespeicherten Rohtext die relevanten Daten, 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 zu bearbeiten. jQuery ermöglicht also DOM-Abfragen mit einer einfachen Syntax und bietet zudem Methoden, die wichtige Aufgaben mit sehr wenig Code erledigen.[6] Bei der Einbindung von jQuery in den Paragraph Handler hat man sich bei der Angabe der Quelle (src) für die aktuellsten jQuery-Version als Link entschieden. Dies hat den Vorteil, dass man jQuery nicht erst runterladen muss und den lokalen Pfad als Quelle angeben muss. Des weiteren wird so gewährleistet, dass immer die aktuellste jQuery-Version verwendet wird.[7] 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. Zudem wird der Textinhalt der Elemente in der jQuery-Auswahl mit der Methode clone() kopiert und mit der Methode text() abgerufen.[8] 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.[9] Anschließend soll der Textinhalt der Paragraphen

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

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.[10] 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 sortiert 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 das Paragraph-Objekt in ein Array zu speichern, benötigt man zuerst den im sessionStorage gespeicherten 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, wurde die Variable 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 korrekte Uhrzeit im Paragraphen enthalten ist. Der reguläre Ausdruck schreibt die Form vor, die die Uhrzeit aufweisen soll. Der gesamte reguläre Ausdruck setzt sich folgendermaßen zusammen: /([01]|2[0-3]):[0-5]/g

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, 3, 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

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

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). Das g (greedy) am Ende des regulären Ausdrucks stellt sicher, dass hier nicht alle Uhrzeiten im kompletten Text, sondern nur ein Vorkommen geliefert wird.[11]

Um im Paragraphen den Mittel- und Endpunkt zu finden und zu erhalten, verwendete man die slice- bzw. search-Methode. Diese Methoden ermöglichen ein schnelles und einfaches Erhalten der gewünschten Positionen im Paragraphen. Alternativ wäre hier auch die Verwendung von jQuery möglich gewesen. Mit Hilfe der Methode search wird nun nach der im regulären Ausdruck definierten Form im Text gesucht und die Position der Übereinstimmung zurückgegeben.[12] 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.[13]

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. 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 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.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.

[11] https://wiki.selfhtml.org/wiki/JavaScript/Objekte/RegExp, abgerufen am 27.11.16
[12] http://www.w3schools.com/jsref/jsref_search.asp, abgerufen am 27.11.16
[13] http://www.w3schools.com/jsref/jsref_slice_array.asp, abgerufen am 27.11.16

Das Array wird als JavaScript Object Notation (JSON) abgespeichert, um zwischen den HTML-Dokumenten Daten austauschen zu können. JSON hat den Vorteil, dass das Format kompakter ist als HTML/XML.[14] Zudem ermöglicht die Methode stringify(value), dass das Array als String in der Variable arrayJSON abgespeichert werden kann. Darüber hinaus wird diese Variable noch mit der Methode saveParagraphText im sessionStorage abgespeichert.

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

[14] JavaScript und jQuery - Interaktive Websites entwickeln, von Jon Duckett S.374

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