FH-Gießen-Friedberg

THM Stundeplan Parser
Dokumentenstrukturen

Franz Weisflug

Inhaltsverzeichnis

1. Projektvorstellung

Ziel des Projekts war es die HTML Stundenpläne der THM Friedberg, die mit Tabellen umgesetzt sind, zu parsen und in eine einfache leserliche Datenstruktur zu bringen. Als Ausgabe Formate wurden JSON so wie XML gewählt. Weiter musste auch mehrere PDF Dokumente geparst werden um eine Datenvollständigkeit gewährleisten zu können. Im laufe der Fertigstellung wurde auch eine Versionsprüfung und das Zwischenspeichern der geparsten Daten umgesetzt. Eine simple API mit der externe Programme auf geparsten Daten zugreifen können wurde ebenfalls fertiggestellt.

2. Technische Umsetzung

Es wurde sich entschieden eine kleine Webanwendung mittel PHP zu schreiben, um diese für alle verfügbar zu machen. Zum parsen der HTML Dokumente wurde ganon[1] benutzt, das Parsen der PDF Dokumente wurde mit Hilfe von smalot pdfprser[2] umgesetzt.

2.1 ganon

Ganon ist eine simple PHP Library mit Hilfe dessen man objektorientiert auf HTML und XML Dokumente zugreifen. Es vereinfacht das modifizieren des DOM Trees und das finden von Elementen mit CSS3 ähnlichen Abfragen. Es ist sehr ähnlich zu jQuery.

```php
$html = file_get_dom('URL');

$html('p');
$html('.myClass');
$html('#myID');
$html('td:first-child');
$html('table table');
$html('li:odd');
$html('a', 2);

$node->parent;
$node->href;
$node->id;
$node->html();
$node->getInnerText();
$node->getPlainText();
$node->firstChild();
$node->lastChild();
```

```php
$node->id = 'myid';
$node->class = 'myclass';
$node->addAttribute('href', 'www.test.com');
$node->href = 'www.test.com';
$node->href = null;
$node->deleteChild(2);
$node->clear();
$node->setOuterText('<a>New</a>');
$node->setInnerText('<a>New</a>');
$node->setPlainText('New Plain Text');
```

Das linke Codebeispiel zeigt wie man auf Elemente eines HTML Dokumentes zugreifen kann. Das rechte Beispiel illustriert das Modifizieren des Dokumentes bzw. der einzelnen DOM Objekte.

1 https://code.google.com/p/ganon/

2 http://www.pdfparser.org/

2.2 smalot pdfparser

Der smalot pdparser ist eine PHP Library die PDF Dokumente parsen kann und diese in Textform umwandeln kann. Unteranderem gibt es Methoden nur einzelne Seiten oder Meta Daten abzurufen. In folgendem Beispiel wird ein PDF Dokument geparst und zu einem simplen Text konvertiert.

```
$parser = new \Smalot\PdfParser\Parser();
$pdf    = $parser->parseFile(PATH);
$text = $pdf->getText();
```

Unerwünschte Textteile wurden mit Hilfe von Regulären Ausdrücken entfernt, zum Beispiel die Foote und Überschriften auf allen Seiten, unerwünschte Zeilenumbrüche und Zeichen. Siehe folgendes Beispiel.

```
$text = preg_replace('/Kürzel Bezeichnung/', '', $text);
$text = preg_replace('/Seite [0-9]* von *[0-9]* *Stand.*\n/', '', $text);
$text = preg_replace('/\n \n/', "\n", $text);
$text = preg_replace('/-\n/', '', $text);
$text = preg_replace('/\n(^[\(].*[\)])/m', '$1', $text);
```

3. Geparste Daten

Um vollständige Daten für eine Stundenplan zu erhalten ist es nötig die benötigten und relevanten Daten aus mehren Quellen zu beziehen. Problem hierbei sind die Informationen die nur als Kürzel oder zusammengesetzte Kürzel bestehen, dies sind die Dozenten Namen, der Name der Veranstaltung, der Studiengang, das Semester und die Studiengangskennung.

3.1 Simple Daten

Alle Dokumente außer der eigentlich Stundenplan waren sehr simpel zu parsen. Dies war Die Dozentenkürzel PDF, die Fächerbezeichnung PDF, die HTML Tabelle mit Studiengangsbezeichnungen und –kennung und die HTML Tabelle mit allen Stundenplänen.

Diese Daten wurden in simplen Dictionaries gespeichert um einfach mit ihren ID bzw. Kennungen auf den Vollnamen oder ihrer Daten zugreifen zu können. Als Beispiel siehe folgendes Studiengangsdictionary.

```
{
    "AE": "Allgemeine Elektrotechnik",
    "ME": "Mechatronik",
    "BI": "Bahningenieurwesen",
    "MI": "Medieninformatik",
    "EI": "Elektro- und Informationstechnik",
    "MM": "Maschinenbau \/ Mechatronik",
    "FM": "Facility Management",
    "PT": "Physikalische Technik",
    "IC": "Information & Communication Engineering",
    "OB": "Optotechnik und Bildverarbeitung",
    "IK": "Informations- und Kommunikationstechnik",
    "SC": "Supply Chain Management",
    "LG": "Logistik",
    "TI": "Technische Informatik",
    "MA": "Wirtschaftsmathematik",
    "WI": "Wirtschaftsingenieurwesen",
    "MB": "Maschinenbau",
    "WK": "Wirtschaftsinformatik"
}
```

Die Kennung der Stundenpläne musste etwas aufgeschlüsselt werden um alle Informationen aus ihr extrahieren zu können. Die ersten zwei stellen repräsentieren das Studiengangskürzel, an dritter Stelle steht die Studiengangskennung, die vierstelle ist das Semester und die letzte Stelle repräsentiert die Variante.

```
MIB4A
Studiengang           Medieninformatik
Studiengangskennung   Bachelor
Semester              4. Semester
Variante              Gruppe A
```

3.2 Stundenplan

Der Stundenplan war das Dokument das nicht auf simple Weiße geparst werden konnte, da dieser nicht nur eine Simple Liste oder Tabelle ist. Die Daten lagen als mehrfach verschachtelte Tabellen vor. Weiter waren atomare Teile der Daten zum Beispiel mit Font-Tags umschlossen. Außerdem wurden die Tabellenattribute colspan und rowspan benutzt, die die Tabelle in eine nicht simple NxN Matrix wandelt. Das heißt des jede Zeile nicht die gleiche Anzahl an Spalten Elementen haben muss.
Die problematische Aufgabe hier war es diese Daten wieder in eine NxN Matrix zu wandeln um mit Hilfe der Koordinaten den Block einer Veranstaltung herauszufinden. Dafür wurde ein eigener Algorithmus entwickelt der Beispielhaft in der algorithmus.php Datei im Beispielprojekt gezeigt wird.

	M		T		W		T		F		S	
B1	0	0	0	0	0	0	0	0	0	0	0	0
B2	0	0	0	0	0	0	0	0	0	0	0	0
B3	0	0	0	0	0	0	0	0	0	0	0	0
B4	0	0	0	0	0	0	0	0	0	0	0	0
B5	0	0	0	0	0	0	0	0	0	0	0	0
B6	0	0	0	0	0	0	0	0	0	0	0	0
B7	0	0	0	0	0	0	0	0	0	0	0	0

Ein Stundeplan besteht aus 6 Tagen, Montag bis Samstag und hat maximal 7 Blöcke pro Tag. Theoretisch ist es möglich, da colspan bis zu 6 benutzt wird, 6 gleichzeitige Vorlesung pro Block in eine Stundenplan zu haben. Praktisch werden aber nur bis zu zwei gleichzeitige Vorlesungen benutzt, also colspan 6 (eine Vorlesung) und 3 (zwei Vorlesungen). Durch das Format wie die Tabelle vorliegt ist jede Vorlesung mindestens 2 rowspans lang. Das bedeutet, dass die Länge einer Vorlesung, das rowspan Attribut geteilt durch 2 ist. Dies resultiert, wie oben zu sehen, in eine 12x7 Matrix, 6 Tage mit 7 Blöcken und maximal 2 Vorlesungen pro Block.

Mit diesen Regeln lassen sich schon einfache Stundenpläne parsen, die nicht den Folgenden Sonderfall enthalten. Problematische Stundenpläne sind diejenigen mit Elementen an einer Stelle (Spalte) in der sie nicht in der Matrix stehen würden. Das ist zum Beispiel im unten stehenden Bild am Mittwoch im 3. Block der Fall, bei der Vorlesung SG (rote Markierung). Dieses Element steht in der Zeile an erster Stellen aber ist in der Matrix an sechster Stelle (Mittwoch), da durch rowspan verrückt.

<table>
<tr><th>Mo</th><th colspan="3">Di</th><th colspan="2">Mi</th><th>Do</th><th>Fr</th><th>Sa</th></tr>
<tr><td>SA. Hg 1) A3 U110</td><td colspan="2" rowspan="2">BMP. Br 4) B1 211</td><td rowspan="2">AVP. Wz 9) B1 111</td><td></td><td rowspan="2">DVI. Eer 13) B1 311</td><td></td><td></td></tr>
<tr><td rowspan="2">SG. Br 10) A4 014</td><td></td><td></td></tr>
<tr><td></td><td rowspan="3">MSI. Km 5) B1 111</td><td rowspan="2">STI. Wb 6) A3 203</td><td rowspan="3"></td><td rowspan="2">AV. Wz 12) A2 111</td><td rowspan="3">FBA. Lom 14) B1 311</td><td rowspan="2">RNG. LDe 15) A A2 007</td><td rowspan="3"></td></tr>
<tr><td>DIF. LHd 2) A2 U117</td><td rowspan="2">SG&#65533;. LW&#65533; 11) B1 211</td></tr>
<tr><td>DIFP. LHd 3) A2 204</td><td colspan="2">TW. Wgd 7) A1 001</td><td></td><td></td></tr>
<tr><td></td><td colspan="3">TWP. Wgd 8) A4 118</td><td colspan="2"></td><td></td><td></td></tr>
</table>

Der Algorithmus füllt eine 12x7 Matrix mit Nullen. Daraufhin geht dieser durch alle Elemente jeder Zeile. Falls keine Vorlesung gefunden wird, wird dieser Block in der Matrix mit einer 2 versehen (geprüft und leerer Block). Falls eine Vorlesung gefunden wird, werden alle relevanten Daten ermittelt, auf die ich nicht einzeln eingehen möchte. In die Matrix wird daraufhin eingetragen ob sich die Vorlesung alleine in diesem Block Stattfindet und wie lange diese Läuft. Der Block in der eine Vorlesung stattfindet lässt sich einfach aus der momentanen Zeile ermitteln, der Tag jedoch ist etwas problematisch. Es muss geprüft werden welche Elemente in einer Zeile noch frei sind (nicht belegt oder 2) und ob die Vorlesung noch im gefundenem Block Platz hat (eine Vorlesung die alleine im Block stattfindet hat keinen Platz in einem Block in dem schon eine Vorlesung stattfindet, aber eine Vorlesung die sich den Block mit einer anderen teilt hätte platz). In Folgender Tabelle ist das vorherige Beispiel in eine 12x7 Matrix gewandelt worden.

	M		T		W		T		F		S	
B1	SA	SA	BMP	BMP	AVP	2	2	2	2	2	2	2
B2	SA	SA	BMP	BMP	AVP	SG	DVI	DVI	2	2	2	2
B3	2	2	MSI	2	2	SG	DVI	DVI	2	2	2	2
B4	2	2	MSI	STI	SGÜ	AV	FBA	FBA	RNG	RNG	2	2
B5	DIF	DIF	MSI	2	SGÜ	2	FBA	FBA	RNG	RNG	2	2
B6	DIFP	DIFP	TW	TW	SGÜ	2	FBA	FBA	2	2	2	2
B7	2	2	TWP	TWP	2	2	2	2	2	2	2	2

4. JSON und XML

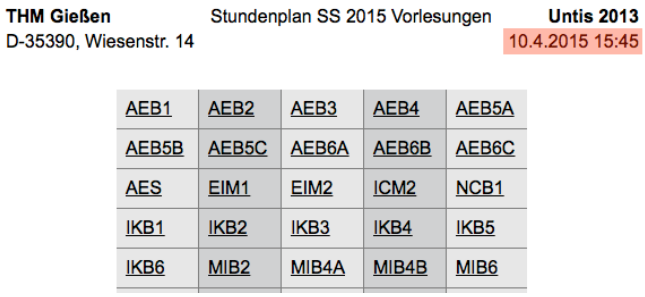
Das sich entschieden wurde alle Daten sowohl in JSON als auch XML zur Verfügung zu stellen musste eine Struktur für diese Daten erstellen werden. Die einfach geparsten Date aus Key Value Paaren bestehen wurden auch so in ein JSON gewandelt. Die XML Dokumente haben einfaches Element und mit geschachtelten Elementen die den Key und die Value repräsentieren. In Folgendem das JSON Beispiel aus Kapitel 3.1 nochmal teilweise als XML

```
<?xml version="1.0"?>
<DegreeList>
    <degree>
        <id>AE</id>
        <name>Allgemeine Elektrotechnik</name>
    </degree>
    <degree>
        <id>ME</id>
        <name>Mechatronik</name>
    </degree>
    <degree>
        <id>BI</id>
        <name>Bahningenieurwesen</name>
    </degree>
    <degree>
        <id>MI</id>
        <name>Medieninformatik</name>
    </degree>

     ...

</DegreeList>
```

Der Stundenplan besteht nur aus Values im Element erster Ordnung und hat deswegen keinen Key oder ID. Alle Daten die geparst werden sind über die API zugänglich. Die Struktur jedes Dokuments können auf der Startseite (index.php) des Projektes angeschaut werden.

5. Performance

Die erste Implementation des Parses war nach ersten Tests noch sehr langsam. Zwischen 7 und 9 Sekunden dauerte es bis eine Anfrage beantwortet werden konnte. Wurden die Dokumentabrufe von https auf http URLs umgestellt, verringerte sich die Abfragezeiten auf 4 bis 5 Sekunden, was immer noch zu langsam war.
Deswegen sollen alle geparsten Daten zwischen gespeichert. Zu jedem Dokument das geparst werden muss, sei es PDF oder HTML Dokument, gibt es auf der zugehörigen Seite auch ein Datum wann dieses Dokument das letzte mal aktualisiert wurde (Siehe folgendes Bild als Beispiel für Stundenplan Aktualisierungsdatum, rote Markierung).



Jedes mal wenn ein Dokument geparst und gespeichert wird, wird auch gleichzeitig dieses Datum gespeichert. So kann man überprüfen ob ein Dokument geparst werden muss oder einfach das vorhandene benutzt werden kann. Dieses Zwischenspeichern reduziert abfragen bis auf ca. 0,3 Sekunden.

6. API

Die API ist über die thmtdata.php Datei zugänglich. Ihr können bis 3 Attribute übergeben werden, action, id und type. Als action können folgende Werte übergeben werden teacherlist, leacturelist, degreelist, timetablelist und timetable. Sie geben an welche Daten benötigt werden, respektive die Dozentenliste, Modulliste, Studiengangsliste, Stundenplanliste und ein Stundenplan. Als type kann entweder json oder xml übergeben werden, was die Daten als diesen Typen zurückgibt. Wenn ein Stundenplan abgefragt wird muss zusätzlich auch eine Stundeplan ID angebende werden, diese ID kommt zum Beispiel aus der Stundenplan liste.

```
thmtdata.php?action=<ACTION>&id=<TIMETABLE KEY>&type=<json|xml>
```

7. Ausblick

Umgesetzt wurde alles was ich mir am Anfang vorgenommen hatte. Trotzdem gibt es noch einige Punkte die mir im laufe des Projektes aufgefallen sind die ich gerne umsetzen oder erweitern möchte.
Momentan werden die Daten so gespeichert wie sie auch vorliegen. Idealerweise sollte die Daten aber als zusammenhängende Datenstruktur gespeichert werden, um zum Beispiel auch Informationen zu einzelnen Vorlesungen bekommen zu könne, alle Vorlesungen zu einem Block zu bekommen oder freie Räume zu finden. Weiter wäre es auch von Vorteil den Raumplan zu parsen, da dieser alle Informationen zur verfügen stellt. Wenn zusammenhängende Daten existieren, wäre es auch möglich sich eigene Stundenpläne zusammenzustellen. Außerdem wären direkte Links zu Modulbeschreibungen und den Dozentenseiten sehr hilfreich.