Stundenplan Parser in Haskell

Ein Projekt von Kevin Peters im Fach “SOE - Weitere Programmiersprache“
Wintersemester 2016/2017

Matrikelnummer: 70430220

Inhaltsverzeichnis

Motivation

Der Stundenplan der “Ostfalia Hochschule für angewandte Wissenschaften“ ist über ein Webportal für außenstehende Personen verfügbar. Dabei muss der Benutzer die URL “http://splus.ostfalia.de/” aufrufen und kann danach verschiedene Filter für den Stundenplan anwenden. Zu diesen Filtern gehören:

- Kategorie
- Planversion

Durch die Filterung der Pläne erhält man am Ende einer Auswahl immer einen eindeutigen Stundenplan, der jedoch, obwohl der Inhalt unterschiedlich ist, immer das gleiche Schema aufweist.
Eine weitere Filterung des Stundenplans kann durch die Filterung der Woche durchgeführt werden. Dabei hat der Benutzer die Wahl eine Woche auszuwählen. Diese Wochen orientieren sich an den Kalenderwochen, jedoch gibt es Inkonsistenzen bei der Vergabe von diesen Wochen, da Kalenderwochen auch über die eigentliche Anzahl an Kalenderwoche hinüber gezählt werden.
Durch die immer wiederkehrende Darstellung des Stundenplans kann daraus geschlossen werden, dass der Stundenplan nur von zwei Variablen abhängig ist:

- Auswahl von Kategorie/Planversion
- Wochenbestimmung

Durch diese beiden Variablen kann der Stundenplan seine Daten so anzeigen, wie die Filterung es zulässt.
Durch die immer wiederkehrende Struktur des Stundenplans sollte es sehr einfach sein den Stundenplan zu analysieren und in eine einheitliche Datenstruktur zu bringen. Dabei findet eine Konvertierung von HTML in eine interne Datenstruktur von Haskell statt.

Projekt

Das Projekt wird mit der funktionalen Programmiersprache Haskell umgesetzt. Diese Programmiersprache eignet sich sehr gut, um gleichartige Strukturen in ein Datenstruktur zu bringen. Ebenfalls eignet sich Haskell sehr gut HTML zu parsen, da HTML rekursiv aufgebaut ist und damit ein perfekter Anwendungsfall für Haskell ist.
Zuerst wird jedoch eine Analyse des Stundenplans durchgeführt. Dabei wurde überprüft wie der Stundenplan im Browser angezeigt wird. Dabei stößt man notgedrungen auf das HTTP Protokoll. Dieses sorgt dafür, dass der Stundenplan vom Server an den Benutzer gesendet wird. Um diese HTTP-Traffic zu überwachen wurde in diesem Projekt das Programm Fiddler verwendet. Dies ist ein HTTP Web Debugging Proxy von Telerik, welcher dazu sehr gut geeignet ist, Traffic aufzunehmen und nachzustellen (http://www.telerik.com/fiddler).
Um den Ablauf des HTTP nachzuvollziehen findet man in “Abbildung 1 - HTTP-Protokoll Ablauf” eine Darstellung, wie der allgemeine Ablauf des HTTP-Protokolls abläuft.
Nachdem HTTP verstanden wurde, kann mit der Evaluierung von HTML fortgefahren werden. HTML ist eine Markup Sprache, die von Browsern zur Darstellung von Inhalten genutzt wird. Dabei sieht ein normales HTML-Dokument wie in “Anhang 2 - Grundlayout eines HTML-Dokuments” aus.
Man sieht, dass die Sprache HTML sehr der Sprache XML ähnlich ist. Daher erhält HTML auch seinen rekursiven Aufbau.
Nachdem die Rahmenbedingungen klar gestellt sind, kann nun mit dem Projekt fortgefahren werden. Dabei wird zuerst die HTTP-Request mit Fiddler aufgezeichnet. Dabei sieht die HTTP-Request wie in “Anhang 3 - POST Request an den Stundenplan” aus. Man sieht, dass ein POST-Request mit verschiedenen Header und einem Body übergeben wird. Dabei sind wichtige Daten die URL und der Body der über die POST-Request an den Server übermittelt werden. Nachdem dies herausgefunden wurde, kann man den Request so verändern, dass man wesentlich weniger Header der Request hinzufügen muss. Man erhält einen Request, der aussieht wie in “Anhang 4 - Angepasste Request”.
Mit der angepassten Request gibt der Server jedoch widersprüchliche HTTP Statuscodes zurück. In diesem Fall ist der Statuscode 302, was eine Weiterleitung implizieren soll, jedoch sendet der Server auch in diesem Fall schon den Stundenplan, was eigentlich nicht passieren sollte.
Da dies nun herausgefunden wurde haben wir einen Parameter der den Inhalt des Stundenplans bestimmt:

- Den Indentifier

Der Identifier ist eine eindeutige Textfolge, die einen Kurs an der Hochschule beschreibt. Diese Identifier lassen sich auch über das HTML bestimmen, wurden jedoch in diesem Projekt als Eingabeparameter bestimmt. Als Beispiel für Identifier kann man die Identifier des “Anhang 5 - Beispiel Identifier” hinzunehmen.
Ein weiterer Parameter ist die Wochenbestimmung. Diese Wochenbestimmung wird auch als HTTP-Post Parameter übertragen und besitzt die Form:
weeks={Wochennummer}
Dabei werden in den meisten Fällen die Kalenderwochen verwendet, jedoch gab es zum Beispiel in dem Jahr 2017 Kalenderwochen über 52, da das Semester von einem bis in das andere Jahr dauert.

Zur Durchführung der HTTP-Requests ist es sinnvoll eine Library zu verwenden, um einen einfachen Zugriff zu HTTP Ressourcen zu erhalten. Dabei habe ich mich entschieden die Library “wreq” zu verwenden. Zu der Installation von “wreq” findet man unter dem Kapitel “Installation” eine Anleitung. Ebenfalls wird für den Zugriff auf verschiedene Elemente innerhalb der “wreq”-Datenstrukturen die Library “Control.Lens” benötigt. Auch diese Installation wird unter dem Kapitel “Installation” erläutert.
Unter dem “Anhang 6 - HTTP Post Request mit ‘wreq’” wird gezeigt, wie eine HTTP Request aussehen könnte, jedoch schlägt dieser Code fehl, da wreq einen ByteString als Parameter erwartet. Ein ByteString wird erwartet, da POST Parameter UTF-8-encodbar sein müssen und auch eine Art von I/O darstellen. Daher kann mit dieser Library kein normaler String verwendet werden. Jedoch ist die Konvertierung von einem String in einen ByteString und andersherum nicht schwierig da interne Haskell Libraries gute Funktionalitäten dafür bieten.
Ebenfalls verwendet die Library “wreq” die Datentypen Maybe und Just, da HTTP Requests nicht unbedingt durchgeführt werden müssen, da es viele verschiedene Fehlerquellen geben kann.
Nachdem die Analyse von der Architektur abgeschlossen wurde, kann nun mit der Analyse des HTML-Dokuments angefangen werden. Der Stundenplan ist in einem Tabellenlayout aufgebaut. Zum einem gibt es Tabellenzellen, die leer sind und welche, in denen Kurse zu finden sind (“Anhang 7 - HTML Layout 1” und “Anhang 8 - HTML Layout 2”). In “Anhang 8 - HTML Layout 2” sieht man ebenfalls welche Daten aus dem HTML geparst werden müssen. Diese Daten werden für die Datenstruktur in Haskell verwendet. Jedoch fehlen Angaben zur Woche und dem Wochentag. Die Woche wird über einen Parameter in der POST-Request Anfrage übergeben - muss also durch den Benutzer übergeben werden. Der Wochentag muss durch die Anzahl der geparsten Spalten (in einem Tabellenlayout ein ‘<td></td>’-Element) bestimmt werden. Vereinfacht kann man sagen, dass eine Reihe 15 Minuten Zeit einnimmt (auf der Y-Achse) und eine Spalte ein Wochentag ist. Somit hat ein Tag, der von 7:00 bis 20:00 angezeigt wird 52 Reihen (Dauer 7:00 - 20:00 in Minuten: 780). Durch diese Reihen lassen sich auch die genauen Uhrzeiten eines Kurses bestimmen.
Die einzelnen Abschnitte in dem HTML werden durch die “splitOn” Methode der internen Library ‘Data.List.Split’ gelöst. Zum Beispiel stellt “Anhang 9 - splitOn Beispiel” ein solches Pattern da. Dies wird mit allen HTML-Elementen so durchgeführt.
Ein Problem gab es jedoch bei dem Parsen des HTML’s, da das HTML, welches durch den Stundenplan gestellt wurde, nicht valid ist. Dadurch funktionieren HTML-Parsing Libraries in diesem Anwendungsfall nicht und es musste auf die “splitOn”-Methode zurückgegriffen werden.
Ein weiteres Problem stellt das HTML-Encoding dar, da einige Schriftzeichen speziell für die Darstellung encoded werden. Dies wird unter anderem dazu gebraucht, um HTML sicher zu gestalten und unter anderem verschiedene Injection-Attacken zu unterbinden. Dies stellt jedoch für das Parsing ein Problem dar, da einige Schriftzeichen nicht so angezeigt werden, wie sie es eigentlich sollten. “Anhang 10 - HTML-Encoding” zeigt einige solcher Schriftzeichen. Die Schriftzeichentabelle für HTML-Encoding ist sehr lang und da es in Haskell bislang keine brauchbare Library dazu gibt, habe ich mich entschlossen dieses Feature außen vor zu lassen. Die Library, die es zur Zeit gibt, soll zwar fast komplett sein, jedoch gibt es einige Schriftzeichen, die die Library und somit auch das Programm zum Abstürzen bringen kann.

In “Anhang 11 - Beispielausgabe” sieht man eine Beispielausgabe des Programms. Das Ergebnis ist eine Liste von Sextupeln. In diesen Sextupeln sind die ersten beiden Einträge die Startzeit und die Endzeit des Kurses. Dafür wurde ein Tupel verwendet um eine gute Darstellung von Zeiten zu erhalten und das Wiederverwenden der Zeit möglichst einfach zu machen. Der dritte Eintrag beschreibt den Wochentag, der vierte Eintrag beschreibt den Kurs, der fünfte Eintrag beschreibt den Raum, in dem der Kurs stattfindet, der sechste und schließlich letzte Eintrag gibt den Dozenten des Kurses an.
Um dem Projekt eine Struktur zu geben wurden die Funktionen in verschiedene Module aufgeteilt. Zu diesen Module zählen:

- HttpUtility
- TextUtility
- TimeUtility
- TupleUtility

Diese Module werden dann von dem Hauptmodul des Programms geladen (core.hs).

Installation

Für die Installation des Programms sind die Standardwerkzeuge von Haskell erforderlich. Zu diesen gehören unter anderem:

- GHCI (https://www.haskell.org/platform/)
- Cabal (ebenfalls https://www.haskell.org/platform/)

Danach muss man die Libraries “wreq” und “Control.Lens” installieren. Dazu ruft man eine neue Kommandozeile auf und führt folgende Befehle aus:

```
$ cabal update
```

```
$ cabal install -j --disable-tests wreq
```

```
$ cabal install lens
```

Die verschiedenen Installationen können einige Minuten andauern.
Nachdem man dies getan hat ruft man den GHCI unter dem Ordner “./src” auf. Danach muss “core.hs” in den Interpreter geladen werden.

Benutzung

Das Programm besitzt eine nützliche Funktion, die sich “getSchedule” nennt. Diese erhält zwei Parameter:

- Die Id des Kurses
- Die Woche

Der Aufruf gelingt zum Beispiel mit:

```
getSchedule "SPLUS659BF0" 51
```

Dies sucht den Stundenplan für die Kategorie “B.Sc. - 4. Sem. Software Engineering (I-B-I4-SE)” in der 51. Wochen des Jahres 2016 heraus.
Eine volle Liste mit den Id’s der Kurse der Kategorie Informatik ist in diesem Verzeichnis als .csv beigelegt.

Anhang

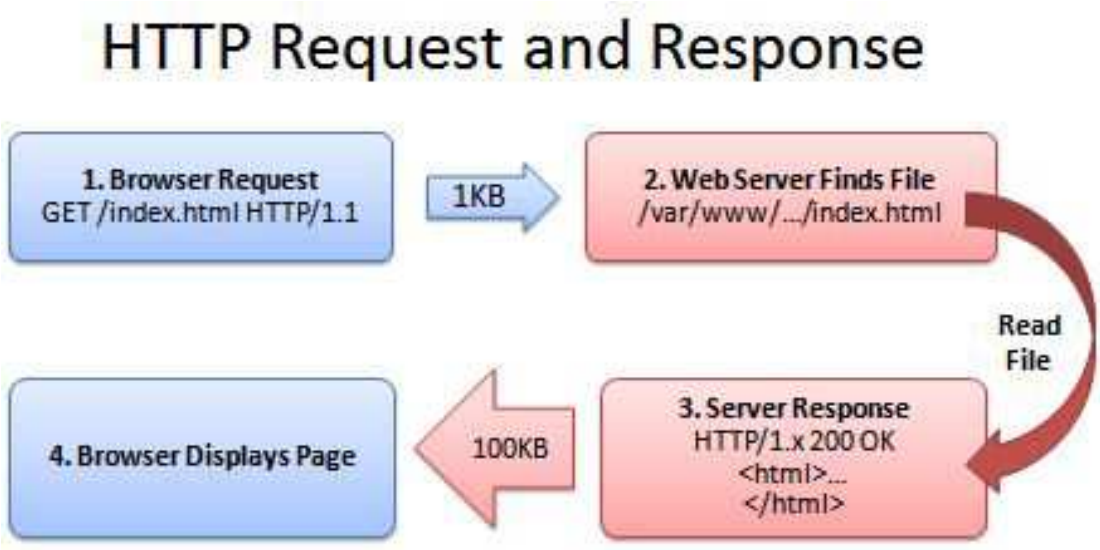

Abbildung 1 - HTTP-Protokoll Ablauf

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Title</title>
  <meta name="description" content="description">
  <meta name="author" content="Kevin Peters">
</head>
<body>
  <div>
    <h1>Heading</h1>
    <p>Paragraph</p>
  </div>
</body>
</html>
```

Anhang 2 - Grundlayout eines HTML-Dokuments

```
POST
http://splus.ostfalia.de/semesterplan123.php?id=1362F014835FFFD0F67159E302EC
```

```
1A3C&identifier=%23SPLUS659BF0 HTTP/1.1
Host: splus.ostfalia.de
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101
Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Referer:
http://splus.ostfalia.de/semesterplan123.php?id=1362F014835FFFD0F67159E302EC
1A3C&identifier=%23SPLUS659BF0
Cookie: PHPSESSID=xxxxxxxxxxxxxxxxxxxxxxxxx
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 31

identifier%5B%5D=%23SPLUSB3BC2D
```

Anhang 3 - POST Request an den Stundenplan

```
POST  http://splus.ostfalia.de/semesterplan123.php?identifier=%23SPLUS659BF0
HTTP/1.1

identifier%5B%5D=%23SPLUSB3BC2D
```

Anhang 4 - Angepasste Request

Id	Vertiefungsrichtung/Jahr
SPLUSB3BC20	B.Sc. - 2. Sem. Software Engineering (I-B-I2-SE)
SPLUS8677CD	B.Sc. - 4. Sem. Information Engineering (I-B-I4-IE)
SPLUS73FFC6	B.Sc. - 4. Sem. Medieninformatik (I-B-I4-MI)
SPLUS659BF0	B.Sc. - 4. Sem. Software Engineering (I-B-I4-SE)
SPLUS69C0D0	B.Sc. - 4. Sem. System Engineering (I-B-I4-SysE)

Anhang 5 - Beispiel Identifier

```
import Control.Lens
import Network.Wreq

standardHeader :: String
standardHeader = "identifier%5B%5D"

standardValue :: String
standardValue = "%23SPLUS659BF0"

postRequest id = do
    response <- post
("http://splus.ostfalia.de/semesterplan123.php?identifier=%23" ++ id)
[(standardHeader) := (standardValue)]
    putStrLn "Done"
```

Anhang 6 - HTTP Post Request mit “wreq”

```
<tr>
  <td rowspan='1' class='row-label-one'>12:15</td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
</tr>
<tr>
  <td rowspan='1' class='row-label-one'>12:30</td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
</tr>
```

Anhang 7 - HTML Layout 1

```
<tr>
  <td rowspan='1' class='row-label-one'>12:00</td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
  <td class='object-cell-border' colspan='1' rowspan='6'>
    <!-- START OBJECT-CELL -->
    <table class='object-cell-args' cellspacing='0' border='0' width='100%'>
      <col class='object-cell-0-1' />
      <tr>
        <td align='center'>MA-SoftwareEngineeringProjekt</td>
      </tr>
    </table>
    <table class='object-cell-args' cellspacing='0' border='0' width='100%'>
      <col class='object-cell-1-1' />
      <tr>
        <td align='center'></td>
      </tr>
    </table>
    <table class='object-cell-args' cellspacing='0' border='0' width='100%'>
      <col class='object-cell-2-0' />
      <col class='object-cell-2-2' />
      <tr>
        <td align='left'>H&ouml;rsaal 127</td>
        <td align='right'>Prof. Dr. B. M&uuml;ller</td>
      </tr>
    </table>
    <!-- END OBJECT-CELL -->
  </td>
  <td class='cell-border'> </td>
  <td class='cell-border'> </td>
</tr>
```

Anhang 8 - HTML Layout 2

```
getCourseName :: String -> String
getCourseName input = head (splitOn "</td>" (splitOn "<td align='center'>"
input !! 1))
```

Anhang 9 - splitOn Beispiel

HTML	Ascii
ö	ö
ü	ü
Hörsaal 127	Hörsaal 127
Prof. Dr. B. Müller	Prof. Dr. B. Müller

Anhang 10 - HTML-Encoding

```
[
((8,15),(9,45),"Thursday","IESE-IT-Sicherheit","H&ouml;rsaal
223","Prof. Dr. S. Gharaei"),
((9,0),(12,0),"Wednesday","SOE-Weitere Programmiersprache -
Projektvortr&auml;ge","H&ouml;rsaal 223","Prof. Dr. M. Huhn"),
((10,0),(11,30),"Wednesday","IESE-IT-Sicherheit","H&ouml;rsaal
223","Prof. Dr. S. Gharaei"),
((12,0),(13,30),"Thursday","SOE-Weitere
Programmiersprache","H&ouml;rsaal 026","Prof. Dr. M. Huhn"),
((12,0),(13,30),"Friday","SOE-SeProjekt","Seminarraum 152","Prof. Dr.
W. Pekrun"),
((14,15),(15,45),"Thursday","SOE-Weitere
Programmiersprache","H&ouml;rsaal 026","Prof. Dr. M. Huhn"),
((14,15),(15,45),"Friday","SOE-SeProjekt","Seminarraum 152","Prof. Dr.
W. Pekrun"),
((16,0),(17,30),"Wednesday","SOE-Fortgeschr. Themen
Softwaretechnik","H&ouml;rsaal 026","Prof. Dr. B. M&uuml;ller"),
((16,0),(17,30),"Thursday","SOE-Fortgeschr. Themen
Softwaretechnik","H&ouml;rsaal 026","Prof. Dr. B. M&uuml;ller")
]
```

Anhang 11 - Beispielausgabe