Verarbeitung von JSON-Daten mit aktuellen XSLT-Techniken

tekom-Jahrestagung 2018 – Stuttgart, 14. November

Dr. Thomas Meinike

Hochschule Merseburg | FB Wirtschaftswissenschaften und Informationswissenschaften
hs-merseburg.de/ww | kiw.hs-merseburg.de | datenverdrahten.de
thomas.meinike@hs-merseburg.de

Motivation 1/3

➔ Warum sollte man XSLT lernen oder Kenntnisse vertiefen?
Ein guter Grund: es könnte ja mal jemand danach fragen!

➔ Mit geeigneten Techniken könnte auch dieses Problem gelöst werden.

HOME HOCHSCHULE MERSEBURG University of Applied Sciences

Motivation 2/3

- **XSLT 3.0** und speziell **XPath 3.1** – beide wurden 2017 final vom W3C verabschiedet – bringen erweiterte Funktionalitäten u. a. zur Abfrage und Verarbeitung von JSON-Daten mit.

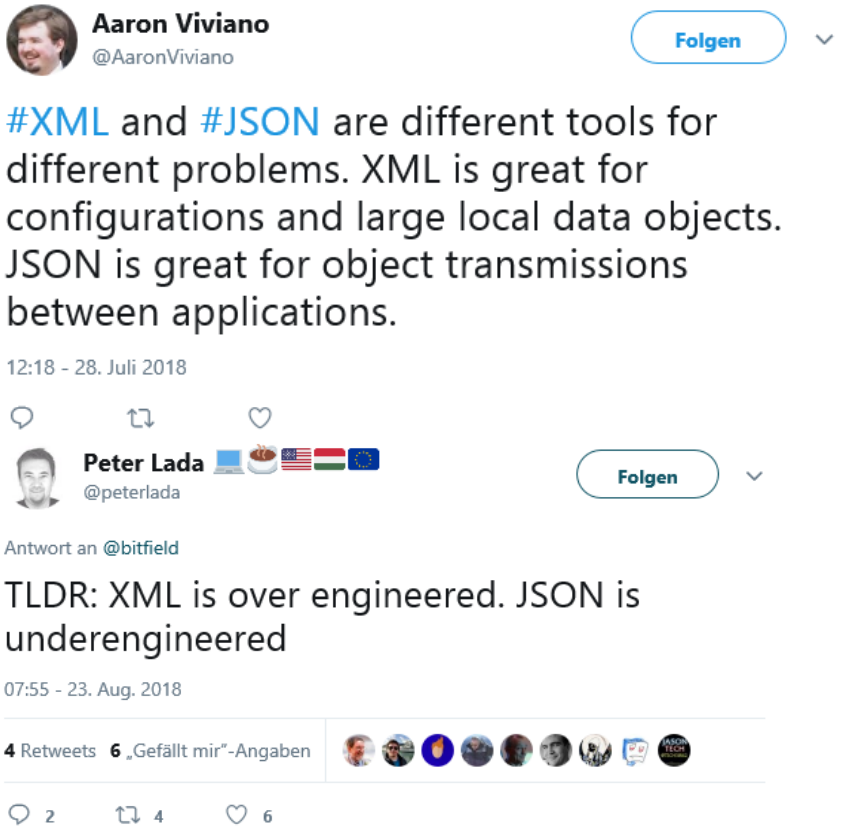
- Organisationen und Verwaltungen stellen Datensätze als Open Data zur Verfügung, wahlweise abgelegt als CSV, XML oder JSON, govdata.de. Vom W3C stammt der Ansatz JSON-LD (Linked Data), json-ld.org.

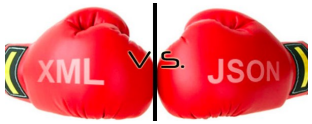
- Auch Konfigurationsdaten können als JSON vorliegen und ggf. additiv zu anderen Formaten in Prozesse integriert werden.

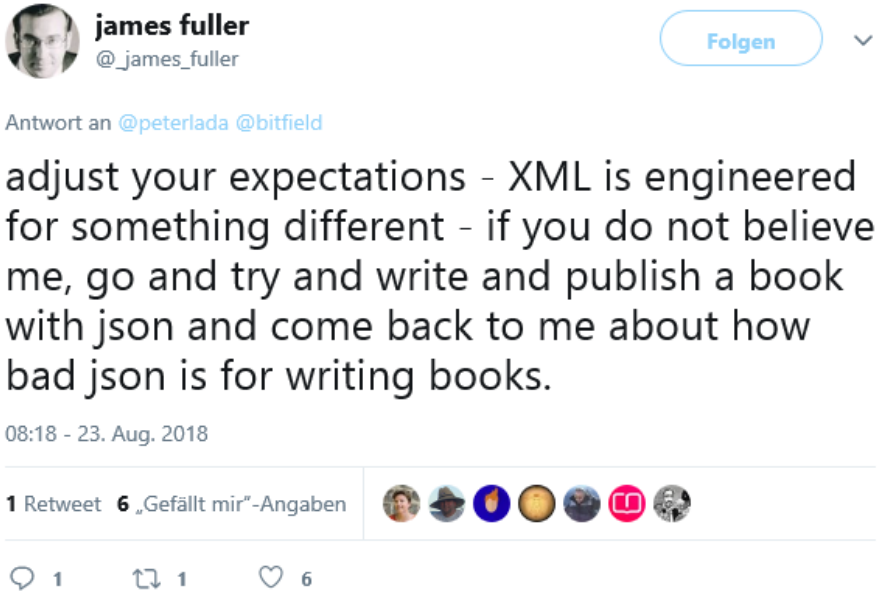
- Codefragmente und eine auf der Basis von Saxon-JS entwickelte Web-Anwendung sollen die Verarbeitung von JSON-Daten veranschaulichen.

Motivation 3/3

➔ Abgrenzung: JSON ist ein überwiegend datenorientierter Ansatz und XML eignet sich besser für stärker dokumentgeprägte Anwendungen:







HOME HOCHSCHULE MERSEBURG University of Applied Sciences

Was ist JSON? 1/2

- JavaScript Object Notation, 2001 von Douglas Crockford entworfen und Standard ECMA-404 »The JSON Data Interchange Syntax«, json.org.

- Datenkodierung erfolgt mit den aus JavaScript bekannten Konstrukten: Object {…}, Array […], Zeichenketten in Anführungszeichen (String), Zahlenwerten als ganze oder Fließkommazahlen (Number) und den Wahrheitswerten true bzw. false (Boolean) sowie null (Nullwert).

- Datenzuweisung erfolgt in Form von Key-Value-Paaren: "key": "value" und diese lassen sich weiter verschachteln.

- JSON Schema ermöglicht Modellierung und Validierung (Draft-Status) z. B. direkt in Code-Editoren, json-schema.org.

Was ist JSON? 2/2

➔ Beispiel zur Ablage von Konferenzdaten (konferenzen.json):

```
{
  "konferenzen": [
    {
      "kid": "k01",
      "name": "tekom-Jahrestagung 2018",
      "ort": "Messe Stuttgart",
      "tage": 3,
      "datum": ["2018-11-13", "2018-11-14", "2018-11-15"],
      "infos": "https://tagungen.tekom.de/",
      "topevent": true
    },
    { weitere Einträge … }
  ]
}
```

Array […] mit Objekten {…}

Datenpaare, dabei ist datum ein weiteres Array […]

3 und true sind keine Strings, stehen somit nicht in "…"!

Äußere Objekthülle

JSON-Techniken in XSLT/XPath 1/8

➔ Grundkonzept **Array**, vgl. XPath Data Model (XDM) <output>

➔ Konstruktor: array{…, …, …, usw.} oder mit […, …, …, usw.]

➔ Zugriff: direkt über den Laufindex ab 1 oder mit Array-Funktionen

```
<xsl:variable name="arr" select="[1, 3.14, 20, 'Text']" as="array(*)"/>

<out>{$arr(1)}</out>              <!-- 1 -->
<out>{$arr(2)}</out>              <!-- 3.14 -->
<out>{array:get($arr, 4)}</out> <!-- Text -->
<out>{array:size($arr)}</out>     <!-- 4 -->
<out>{$arr?3}</out>               <!-- 20 | mit Lookup-Operator ? -->
```

➔ Funktionen: array:append, filter, flatten, fold-left, fold-right, for-each, for-each-pair, **get**, head, insert-before, join, put, remove, reverse, **size**, sort, subarray, tail

JSON-Techniken in XSLT/XPath 2/8

- Grundkonzept **Map**, vgl. XPath Data Model (XDM)
- Konstruktor: map{…, …, …, usw.}
- Zugriff: direkt über Keys, Indizes oder mit Map-Funktionen

```
<xsl:variable name="map" as="map(*)"
              select="map{'test': ['abc', 123, 'http://example.com/']}"/>

<out>{$map('test')(1)}</out>          <!-- abc -->
<out>{$map?test?2}</out>              <!-- 123 -->
<out>{map:get($map, 'test')?3}</out> <!-- http://example.com/ -->
<out>{map:size($map)}</out>           <!-- 1 -->
<out>{array:size($map?test)}</out>    <!-- 3 -->
```

- Funktionen: map:contains, entry, find, for-each, **get**, keys, merge, put, remove, **size**

JSON-Techniken in XSLT/XPath 3/8

➔ Einlesen von JSON-Daten aus Dateien mit fn:json-doc():

```
<xsl:variable name="json" select="fn:json-doc('konferenzen.json')" as="map(*)"/>
<xsl:variable name="daten" select="$json?konferenzen" as="array(*)"/>
```

➔ Weitere Abfrage der vorhandenen Unterstrukturen, hier Array-Zugriff auf die vom Konferenzen-Objekt abgeleitete Array-Variable:

```
<out>{array:size($daten)}</out> <!-- 1 -->
<out>{$daten(1)('name')}</out>  <!-- tekom-Jahrestagung 2018 -->
<out>{$daten?1?ort}</out>       <!-- Messe Stuttgart -->
<out>{$daten?1?datum(2)}</out>  <!-- 2018-11-14 -->
<out>{$daten?1?datum?3}</out>   <!-- 2018-11-15 -->
```

➔ Nutzung des Lookup-Operators **?** vereinfacht die Abfragen, ermöglicht später sehr komfortable Nutzung auch mit Konstrukten wie ?*?[…]

JSON-Techniken in XSLT/XPath 4/8

➔ Eingelesene Daten erweitern, eine Konferenz hinzufügen:

```
<xsl:variable name="daten_neu" select="array:append($daten, map{
  'kid': 'k02',
  'name': 'XML Prague 2019',
  'ort': 'University of Economics Prague',
  'tage': 3,
  'datum': ['2019-02-07', '2019-02-08', '2019-02-09'],
  'infos': 'http://www.xmlprague.cz/',
  'topevent': true()
  })" as="array(*)"/>
```

➔ Prüfung der erfolgten Erweiterung:

```
<out>{array:size($daten_neu)}</out>  <!-- 2 -->
<out>{map:size($daten_neu(2))}</out> <!-- 7 -->
<out>{$daten_neu?2?name}</out>       <!-- XML Prague 2019 -->
```

JSON-Techniken in XSLT/XPath 5/8

➔ Selektive Abfrage nach einem eindeutigen Kriterium (hier kid):

```
<xsl:variable name="abfrage_1" as="map(*)"
              select="for $i in 1 to array:size($daten_neu)
                      return $daten_neu($i)[?kid = 'k02']"/>

<out>{$abfrage_1?ort}</out> <!-- University of Economics Prague -->
```

➔ Abfrage-Kurzform:

```
<xsl:variable name="abfrage_2" as="map(*)"
              select="$daten_neu?*[?kid = 'k02']"/>

<out>{$abfrage_2?ort}</out> <!-- University of Economics Prague -->
```

JSON-Techniken in XSLT/XPath 6/8

➔ Daten mit fn:serialize() serialisieren und in Ausgabedatei schreiben:

```
<xsl:variable name="json_neu" select="map{'konferenzen': $daten_neu}" as="map(*)"/>

<xsl:result-document href="output.json">
  <xsl:copy-of select="fn:serialize($json_neu, $options)"/>
</xsl:result-document>
```

```
{
 "konferenzen": [
    {
     "name":"tekom-Jahrestagung 2018",
     "...": "..."
     },
    {
     "name":"XML Prague 2019",
     "...": "..."
    }
  ]
}
```

➔ Ausgabeoptionen in *$options*:

```
<xsl:variable name="options" as="map(*)"
              select="map{
                'encoding': 'UTF-8',
                'method': 'json',
                'indent': true(),
                'use-character-maps': map{'/': '/'}
              }"/>
```

<------ vermeidet \/

JSON-Techniken in XSLT/XPath 7/8

➔ JSON nach XML wandeln mit fn:json-to-xml():

– Eingabe:

```
{"test": ["abc", 123, "http://example.com/"]}
```

– Ausgabe:

```
<map xmlns="http://www.w3.org/2005/xpath-functions">
  <array key="test">
    <string>abc</string>
    <number>123</number>
    <string>http://example.com/</string>
  </array>
</map>
```

➔ fn:xml-to-json() erzeugt umgekehrt aus XML einen JSON-String und fn:parse-json() erzeugt aus einem JSON-String Maps bzw. Arrays.

JSON-Techniken in XSLT/XPath 8/8

➔ Alternative Map-Deklaration mit xsl:map:

```
<xsl:variable name="datensatz" as="map(*)">
  <xsl:map>
    <xsl:map-entry key="'kid'" select="'k02'"/>
    <xsl:map-entry key="'name'" select="'XML Prague 2019'"/>
    <xsl:map-entry key="'ort'" select="'University of Economics Prague'"/>
    <xsl:map-entry key="'tage'" select="3"/>
    <xsl:map-entry key="'datum'" select="['2019-02-07', '2019-02-08', '2019-02-09']"/>
    <xsl:map-entry key="'infos'" select="'http://www.xmlprague.cz/'"/>
    <xsl:map-entry key="'topevent'" select="true()"/>
  </xsl:map>
</xsl:variable>

<xsl:variable name="daten_neu" select="array:append($daten, $datensatz)" as="array(*)"/>
```

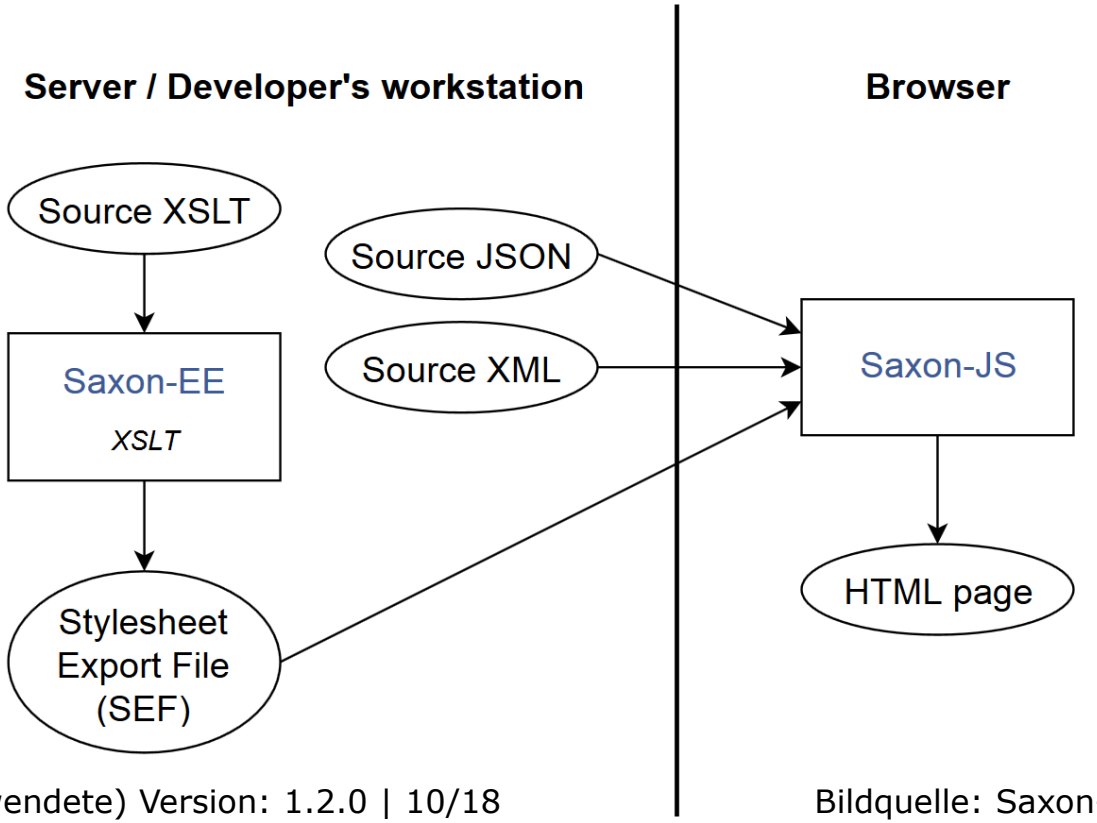
➔ Kindelemente xsl:map-entry mit Attributen key und select für die Zuweisung der Daten.

```
</output>
```

Praxisbeispiel mit Saxon-JS 1/9

➔ Saxon-JS ermöglicht die Nutzung von XSLT 3.0 im Browser und kann sowohl XML- als auch JSON-Daten verarbeiten (→ #tekom17-Vortrag).

➔ Prinzip:

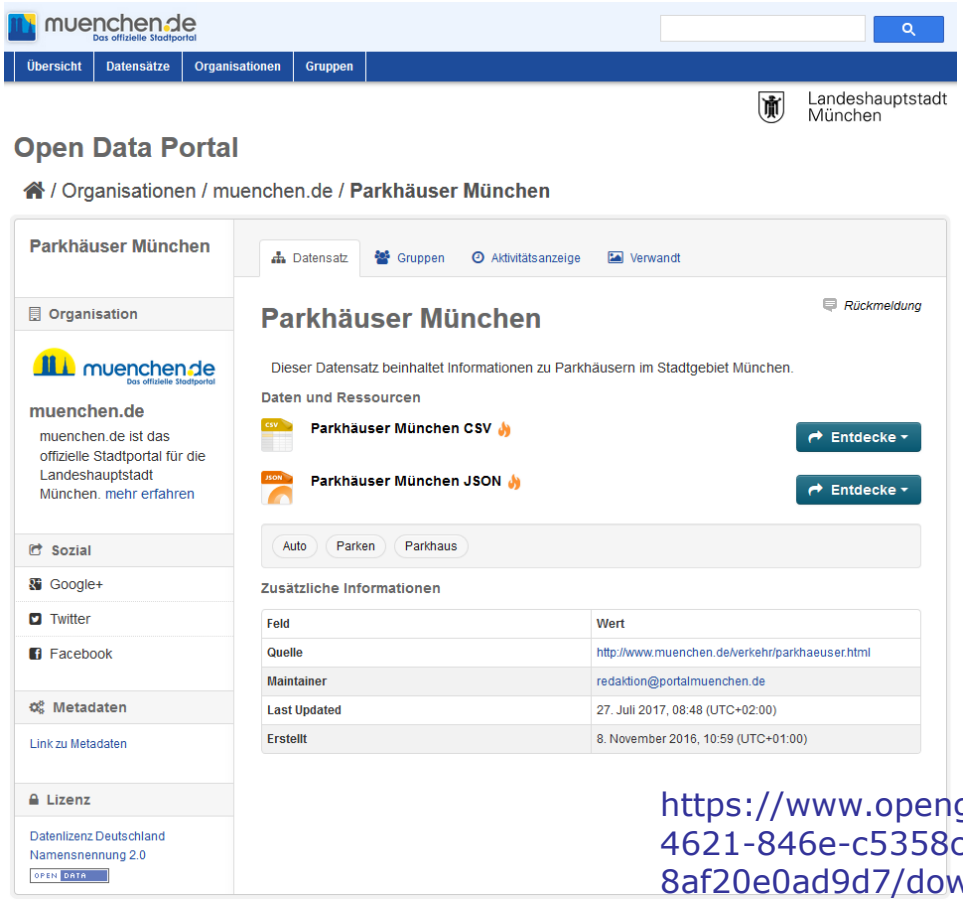


Aktuelle (verwendete) Version: 1.2.0 | 10/18

Bildquelle: Saxon-JS-Dokumentation

Praxisbeispiel mit Saxon-JS 2/9

➔ Verwendet wird ein JSON-Datensatz mit den Parkhäusern in München:



https://www.opengov-muenchen.de/dataset/addaa7d4-40be-4621-846e-c5358cbe3f26/resource/76554a21-34bc-4f62-b609-8af20e0ad9d7/download/2017-07-27parkhaeuser-muenchen.json

Praxisbeispiel mit Saxon-JS 3/9

➔ JSON-Struktur (72 Datensätze unterhalb von "places"):


```
{
  "places": [
    {...}, {
      "id": "121096",
      "title": "Marienplatz Großgarage",
      "city": "München",
      "street": "Rindermarkt",
      "streetNumber": "16",
      "location": {
        "lat": "48.136227",
        "lng": "11.574723"
      },
      "media": {"listPictureUrl":
        "http://images.portal.muenchen.de/[...]/listing-parkgarage-marienplatz.jpg"},
      "isProduct": false,
      "detailUrl": "https://muenchenapis.de/places/api/v2/places/121096.json",
      "urlInPortal": "http://www.muenchen.de/verkehr/orte/121096.html",
      "ratingAverage": "2.30",
      "allowPhotoUpload": true
    }, {...}
  ]
}
```

Praxisbeispiel mit Saxon-JS 4/9

➔ Vorgehen: Daten mit fn:json-doc() in Map-Variable *$json* einlesen und *$places*-Array abrufen:

```
<xsl:variable name="json" select="fn:json-doc('...')" as="map(*)"/>
<xsl:variable name="places" select="$json?places" as="array(*)"/>
```

➔ Anzahl Parkhäuser ermitteln und ausgeben:

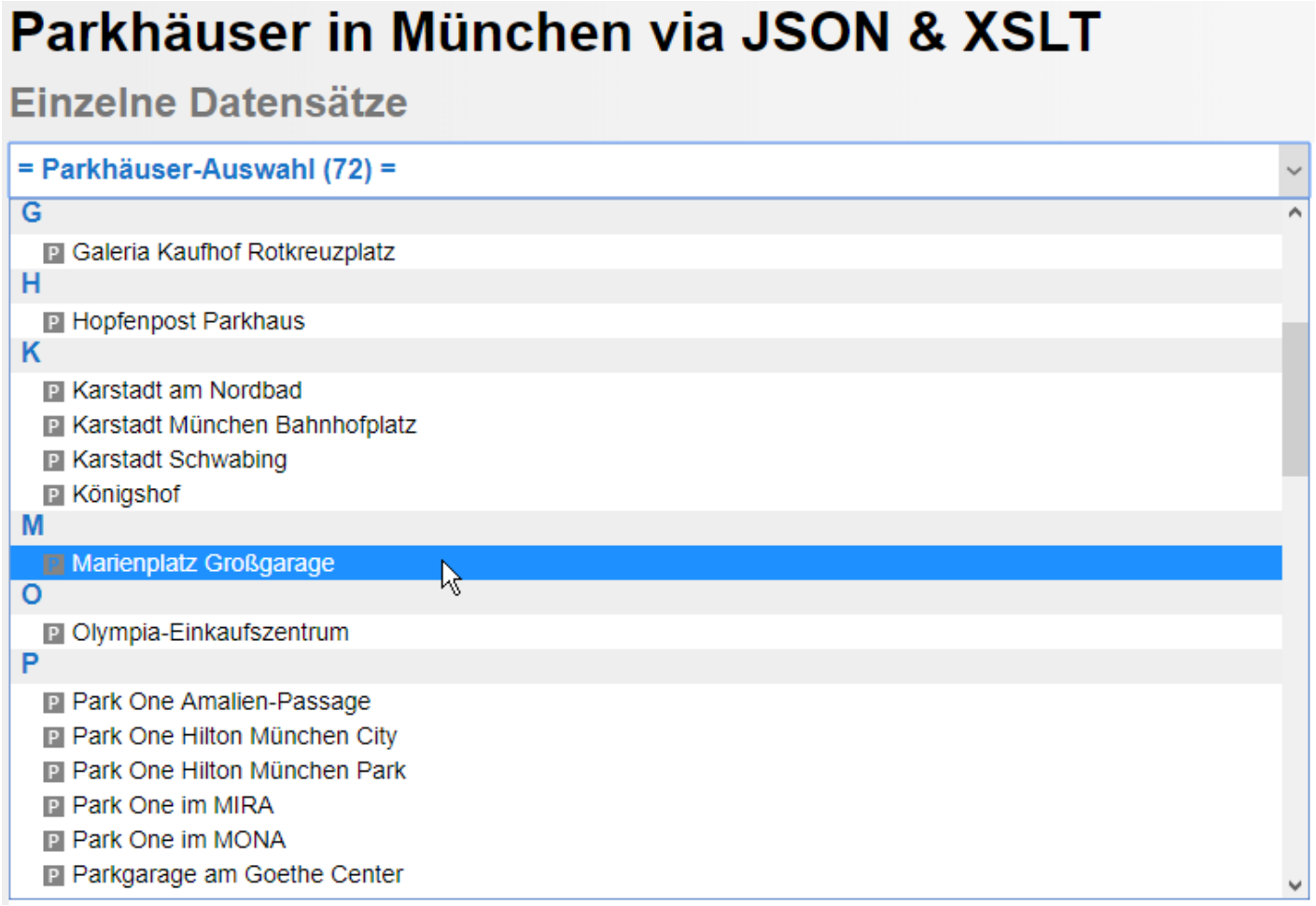
```
<xsl:variable name="anzahl" select="array:size($places)" as="xs:integer"/>
```

➔ Sortierte und gruppierte Auswahlliste aufbauen:

```
<select id="parkhaus">
  <option value="info">= Parkhäuser-Auswahl ({$anzahl}) =</option>
  <!-- ... -->
  <xsl:for-each select="fn:current-group()">
    <option value="{$place?id}|{$pos}">{$place?title}</option>
  </xsl:for-each>
  <!-- ... -->
</select>
```

Praxisbeispiel mit Saxon-JS 5/9

➔ Ergebnis:



Praxisbeispiel mit Saxon-JS 6/9

➔ Parkhaus auswählen und ausgewählte Daten anzeigen:

```
<xsl:template match="select[@id eq 'parkhaus']" mode="ixsl:onchange">
  <xsl:result-document href="#ergebnis" method="ixsl:replace-content">
    <xsl:variable name="curopt" select="." as="element()"/>
    <xsl:variable name="curval" select="ixsl:get($curopt, 'value')" as="xs:string"/>
    <xsl:variable name="curpid" select="fn:substring-before($curval, '|')" as="xs:string"/>
    <xsl:variable name="curpos" select="fn:substring-after($curval, '|')" as="xs:string"/>
    <xsl:variable name="curplc" select="$places?*[?id = $curpid]" as="map(*)"/>
    <table>
      <!-- ... -->
      <td>{$curpos}</td>
      <td>{$curpid}</td>
      <td>{$curplc?street || ' ' || $curplc?streetNumber}</td>
      <td><img src="{$curplc?media?listPictureUrl}" alt="Parkhaus" /></td>
      <td>{$curplc?ratingAverage}</td>
      <td><a href="{$curplc?urlInPortal}" target="details">···</a></td>
      <!-- ... -->
    </table>
  </xsl:result-document>
</xsl:template>
```

Marienplatz Großgarage

Nr.	ID	Straße	Bild	Rating	Details
M1	121096	Rindermarkt 16		2.30	···

➔ Ergebnis:

Praxisbeispiel mit Saxon-JS 7/9

➔ Geokoordinaten auf OpenStreetMap-Karte abbilden:

```
<iframe src="https://www.openstreetmap.org/export/embed.html?bbox={$bbox}
  &amp;marker={$lat},{$lon}&amp;layer=mapnik" width="{$width}" height="{$height}">
  <!-- Alternativlink -->
</iframe>
```

➔ Nötige Parameter werden abgefragt bzw. vorgegeben:

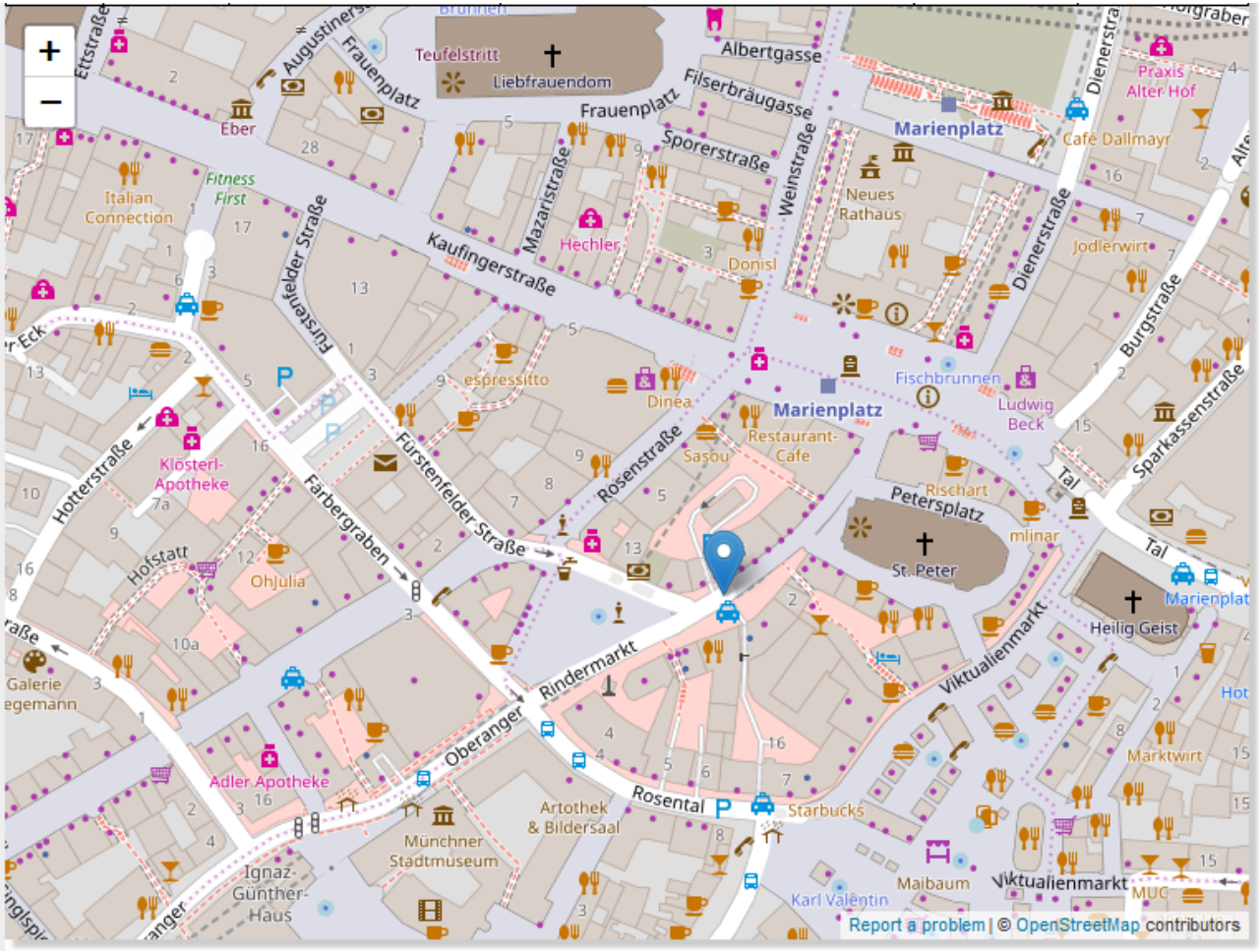
```
<xsl:variable name="lat" select="xs:double($curplc?location?lat)" as="xs:double"/>
<xsl:variable name="lon" select="xs:double($curplc?location?lng)" as="xs:double"/>
<xsl:variable name="zoom" select="17" as="xs:integer"/>
<xsl:variable name="width" select="752" as="xs:integer"/>
<xsl:variable name="height" select="564" as="xs:integer"/>
```

➔ Bounding-Box wird nach speziellem OSM-Algorithmus mit separater XSLT-Funktion *tm:getBBox()* berechnet:

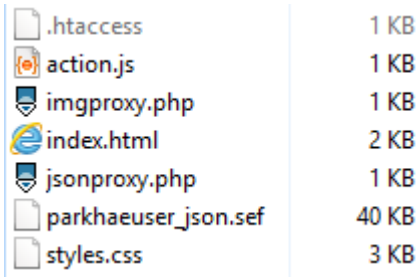
```
<xsl:variable name="bbox" select="tm:getBBox($lat, $lon, $zoom, $width, $height)"
                          as="xs:string"/>
```

Praxisbeispiel mit Saxon-JS 8/9

➔ Ergebnis:



Praxisbeispiel mit Saxon-JS 9/9



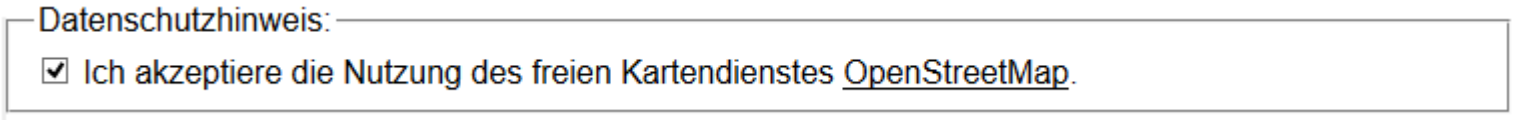
➔ Weitere technische Details:

- JSON-Daten und Bilder werden über Proxy-PHP-Scripts geladen, um Probleme mit Security-Policies, etwa HTTPS vs. HTTP auf den beteiligten Servern zu umgehen (siehe Code).

- OSM-Karte im IFrame benötigte ebenfalls Anpassungen (.htaccess):

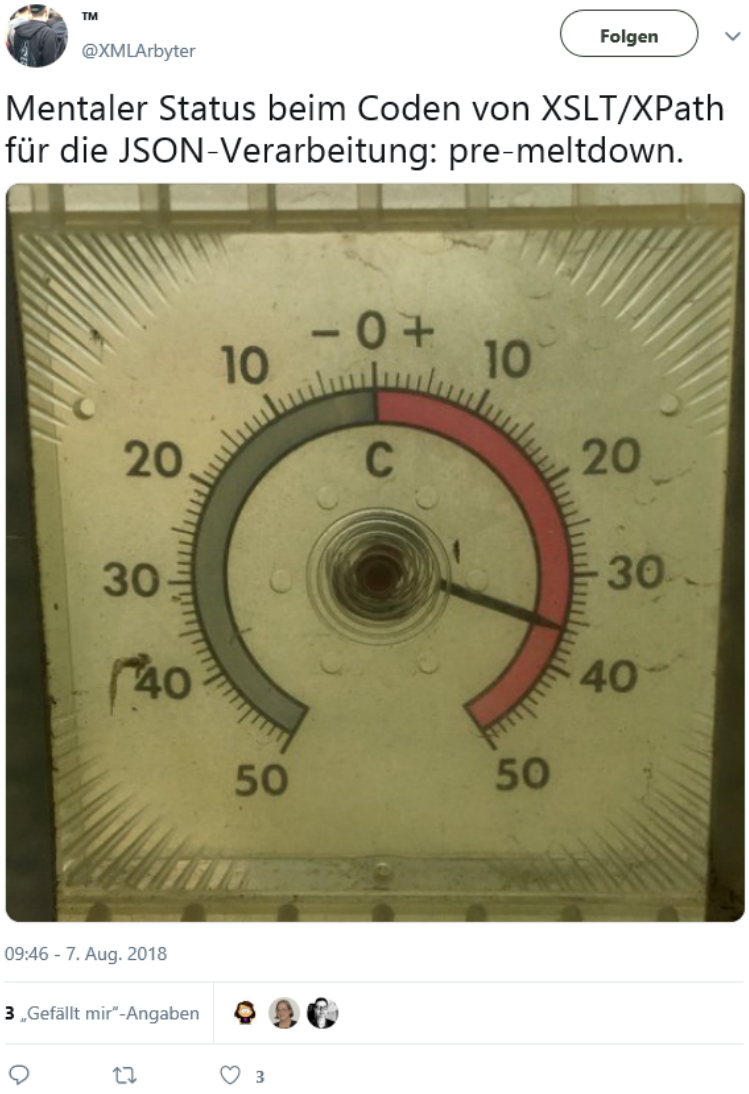
```
Header set Content-Security-Policy "frame-src https://www.openstreetmap.org"
Header set X-Content-Security-Policy "frame-src https://www.openstreetmap.org"
Header set X-WebKit-CSP "frame-src https://www.openstreetmap.org"
```

- Für den Zugriff auf OSM wurde eine Datenschutz-Abfrage eingebaut:



Fazit und Ausblick 1/2

- JSON-Techniken erweitern das Methodenspektrum zur Verarbeitung strukturierter Daten.
- Auch bei der Transformation von XML lassen sich speziell die neuen Konzepte für Arrays und Maps sinnvoll einsetzen.
- Hier nicht thematisiert:
  XPath 3.1 bildet ebenfalls die Grundlage zur Nutzung der Technologie XQuery.



Fazit und Ausblick 2/2

➔ Es bleibt spannend …

Via Matt Owen – https://twitter.com/Jaruzel/ (Thanks for sharing!)

Literatur und Ressourcen 1/2

Birnbaum, D. J.: What's new in XSLT 3.0 and XPath 3.1?
http://dh.obdurodon.org/xslt3.xhtml

Cagle, K.: Why You Should Be Using XSLT 3.0.
https://www.xml.com/articles/2017/02/14/why-you-should-be-using-xslt-30/

ECMA: Standard ECMA-404 – The JSON Data Interchange Syntax.
https://www.ecma-international.org/publications/standards/Ecma-404.htm

GovData: Das Datenportal für Deutschland. https://www.govdata.de/

Grupe, W.: XML – Technologien | Grundlagen | Validierung | Auswertung.
mitp-Verlag 2018. Online-Material: http://www.wilfried-grupe.de/

Introducing JSON: https://json.org/

JSON-LD: JSON for Linking Data. https://json-ld.org/
+ **W3C Working Group**: https://www.w3.org/2018/json-ld-wg/

JSON Schema: https://json-schema.org/

Literatur und Ressourcen 2/2

Meinike, T.: Interaktive XML-Verarbeitung im Browser mit Saxon-JS und XSLT 3.0. In: tekom, Gesellschaft für technische Kommunikation e. V., Tagungsband zur Jahrestagung 2017, S. 177-180. https://datenverdrahten.de/PDF/tekom2017_IN29_Meinike.pdf

Meinike, T.: Parkhäuser in München via JSON und XSLT. https://datenverdrahten.de/xslt3/saxon-js/parkhaeuser/

OpenData-Portal der Landeshauptstadt München: Parkhäuser München. https://www.opengov-muenchen.de/dataset/parkhaeuser-munchen

OpenStreetMap Foundation: OpenStreetMap. https://www.openstreetmap.org/about

Saxonica: Saxon-JS. https://saxonica.com/html/saxon-js/

W3C: XSL Transformations (XSLT) Version 3.0. https://www.w3.org/TR/xslt-30/

W3C: XML Path Language (XPath) Version 3.1. https://www.w3.org/TR/xpath-31/

W3C: XQuery and XPath Data Model 3.1. http://www.w3.org/TR/xpath-datamodel-31/

Feedback erwünscht …

➔ URL direkt aufrufen (auch nach der Tagung möglich):
https://in22.honestly.de

➔ Oder QR-Code scannen:

Danke für Ihr Interesse!