Jugendstadtplan Leipzig – technische Dokumentation

Leipzig Data Projekt

Version vom 18. Februar 2015

1 Hintergrund

Die folgende Beschreibung dokumentiert die Überarbeitung der Kartendarstellung des Jugendstadtplanprojekts, die von einem studentischen Projektteam im Rahmen des Interdisziplinären Moduls „Kreativität und Technik“[1] im Sommersemester 2013 erstellt wurde.

Die ursprüngliche Kartendarstellung wurde von Emanuel Ott (Student im Bachelor Afrikanistik) im Sommersemester 2013 auf der Basis von CloudMade[2] erstellt. CloudMade hat seinen kostenfreien Kartenservice allerdings zum Mai 2014 eingestellt[3].

Ziel der Überarbeitung war es, neben der Herstellung von Konnektivität zu einem anderen Kartenprovider die Website auf der Basis des Web-Frameworks *Bootstrap*[4] neu aufzubauen sowie die Datenschicht über die PHP-Bibliothek *EasyRDF*[5] anzubinden und damit ein prototypisches Beispiel zu schaffen, wie Webseiten mit Kartendarstellungen von geolokalen RDF-basierten Informationen aufgebaut werden können.

Die Überarbeitung wurde durch Immanuel Plath (Student im Master Informatik) im Wintersemester 2014/15 im Rahmen eines Praktikumsauftrags ausgeführt und nachfolgend weiter angepasst.

Mehr zum Jugendstadtplanprojekt siehe `http://www.leipzig-data.de/Jugendstadtplan`.

2 Installation

2.1 Requirements

- Webserver: z.B. Apache, Nginx.
- PHP: mindestens 5.4.
- PHP muss die Rechte besitzen, auf JSON und INI Dateien zugreifen zu können.
- PHP-Paketverwaltung *composer*[6].

[1] `http://bis.informatik.uni-leipzig.de/de/Lehre/Graebe/Inter`

[2] `http://cloudmade.com/`

[3] „In March 2014 we announced that as of May 1st 2014 CloudMade will be discontinuing service of Map Tiles, Geocoding, Routing and Vector Stream Server APIs. We will provide only Map Tiles API for Enterprise users.“ `http://support.cloudmade.com/`

[4] `http://getbootstrap.com/`

[5] `http://www.easyrdf.org/`

[6] `https://getcomposer.org/`

2.2 Installation

- Sourcecode herunterladen und auf Webspace entpacken.
- „composer install“ ausführen (lädt die benötigte Bibliothek EasyRDF).
- Im Webbrowser die Datei „index.php“ im Verzeichnis „public“ aufrufen.

3 Beschreibung der Applikation

3.1 Verzeichnisstruktur der Applikation

- config – Konfigurationsdateien der Applikation.
- data – In diesem Verzeichnis sind alle *Daten* der Applikation abgelegt, also die lokalen RDF-Daten (Verzeichnis „rdf“) und die Übersetzungsdateien (Verzeichnis „translation“).
- module – Dieses Verzeichnis enthält alle funktionalen PHP-Komponenten.
- public – Aufbau der Website. Dieser Verzeichnis ist das einzig, welches von außen erreichbar sein muss.
- vendor – Dieses Verzeichnis enthält alle importierten Bibliotheken bzw. Applikationsabhängigkeiten. Autogeneriert durch Composer.
- composer.json – Composer-Projektbeschreibung.

3.2 Verwendete Bibliotheken

- RDF Framework *EasyRDF* – `http://www.easyrdf.org/`
- Web Framework *Bootstrap 3* – `http://getbootstrap.com`
- Map Framework *Leaflet* – `http://leafletjs.com`
- HTML/CSS Flaggen – `http://lipis.github.io/flag-icon-css`

3.3 Designprinzipien

Kartendarstellung: Für *Kartendarstellungen* ist zu beachten, dass wir auf Karten zugreifen, die im frei im Netz zur Verfügung gestellt werden. Hierfür müssen erhebliche externe Serverkapazität vorgehalten werden, so dass die im Einzelnen genannten „fairen Nutzungsbedingungen“ einzuhalten sind. So heitß es bei *OpenStreetMap:*

> We are in principle happy for our map tiles to be used by external users for creative and unexpected uses – in contrast to most web mapping providers, which insist that you use only their supplied API. ... However, OpenStreetMap's own servers are run entirely on donated resources. They have strictly limited capacity. Heavy use of OSM tiles adversely affects people's ability to edit the map, and is an abuse of the individual donations and sponsorship which provide hardware and

> bandwidth. ... OpenStreetMap data is free for everyone to use. Our tile servers are not. ... But because OpenStreetMap data is free, many other organisations provide map tiles made from OSM data. (Quelle: `http://wiki.openstreetmap.org/wiki/Tile_usage_policy`)

Als Kartenprovider wird in dieser Anwendung nun *Mapbox* (`https://www.mapbox.com`) verwendet, die Interaktion über die JavaScript-Bibliothek *Leaflet* ermöglicht. Der Kartenprovider kann in der Datei `custom-map.js` einfach ausgewechselt werden, wenn der neue Provider ebenfalls *Leaflet* unterstützt.

Webschnittstelle: Die *Webschnittstelle* (Verzeichnis „public“) ist leichtgewichtig gestaltet – in den Dateien `public/index.php` und `public/about.php` werden die jeweiligen Webseiten aus einzelnen Bausteinen zusammengesetzt, die in der Datei `module/html.php` und weiteren Dateien im Verzeichnis „module“ definiert werden. Mehrsprachigkeit wird durch ein Translationskonzept realisiert.

Mehrsprachigkeit: Die Sprache kann über einen GET-Parameter `lang` eingestellt werden, der zu Beginn einer dynamisch aufzubauenden Seite über die Befehle

```
$language = getLanguage();
$translation = getTranslation($language);
```

die Sprache setzt und die entsprechenden Übersetzungen von Schlüsselworten aus einer Sprachdatei im Verzeichnis `data/translations` lädt. Damit ist ein leichtes Editieren und Lesen der Dateien möglich. Um eine weitere Sprache hinzuzufügen, muss lediglich eine der vorhandenen Sprachen in die gewünschte Sprache übersetzt und im selben Verzeichnis abgelegt sowie die neue Sprache in der Funktion `getLanguage()` in `translate.php` und in der Sprachauswahl (`pageNavigation()` in `html.php`) nachgetragen werden.

Beim Aufbau der Seite wird der aktuelle Wert in einem div-Segment

```
<div id="languageStore">de</div>
```

gespeichert, das mit der JS-Funktion `getUserLanguage()` ausgelesen wird, um den dymanisch nachgeladenen Inhalten ebenfalls Zugriff auf die Spracheinstellung zu geben. Über diesen Mechanismus werden sprachspezifische Elemente auch aus den RDF-Graphen ausgelesen.

Ist die angeforderte Sprache nicht verfügbar oder der Parameter ungültig, wird die Website in der Standardsprache (Deutsch) ausgeliefert. Die Standardsprache kann in den Applikationseinstellungen im Verzeichnis „config“ angepasst werden.

Eine Änderung der Sprache (auch über das Sprachauswahlmenü) ist immer mit dem Neuaufbau der Webseite `index.php` verbunden.

3.4 Seitenaufbau mit Bootstrap

Das Bootstrap-Framework ist ein verbreitetes Framework, um responsive Design umzusetzen, bei dem Webseiten ihre Gestalt ändern, wenn die Seitenbreite zu klein wird. Das Framework

integriert außerdem assistive Technologien wie Screenreader, um barrierearme Webseiten gestalten zu können. Mehr zum Bootstrap-Framework siehe `http://holdirbootstrap.de`.

Bootstrap teilt Webseiten in Boxen ein (Container) `<div class="container">`, die in Zeilen `<div class="row">` und jene weiter in Spalten aufgeteilt werden. Responsive Design wird erreicht, indem die Zahl der Spalten an die Auflösung des Geräts angepasst wird, wobei extrem kleine (xs), kleine (sm), mittlere (md) und große (ld) Auflösungen unterschieden werden[7]. Typischerweise wird mit 12 Spalten gerechnet, Elemente können sich über mehrere Spalten erstrecken `<div class="col-sm-4 col-md-4">`. Übersteigt die Summe der Spaltenbreiten 12, so wird eine neue Zeile begonnen bzw. bei Navigationsbalken dieser in ein Dropdown-Menü umgewandelt (Wechsel zwischen minimierter und horizontaler Ansicht)[8].

3.5 Beschreibung der einzelnen PHP-Dateien

index.php: Diese Datei hat die Aufgabe, die Webseite mit der Kartendarstellung auszuliefern. Über den GET-Parameter `lang` kann die Sprache eingestellt werden.

Beispiel: `index.php?lang=de` oder `index.php?lang=en`.

about.php: Diese Datei hat die Aufgabe, die Webseite mit der Projektbeschreibung auszuliefern. Über den GET-Parameter `lang` kann wieder die Sprache eingestellt werden, nach der entschieden wird, ob die Projektbeschreibung aus `about_de.html` oder aus `about_en.html` eingeladen wird.

details.php: Diese Datei wird nur via Ajax-Request aufgerufen. Über den GET-Parameter „uri“ kann ein Ort über dessen URI angefragt werden. Wird der entsprechende Ort im RDF Graphen gefunden, werden die zugehörigen Details zurückgegeben. Über den zweiten Parameter „lang“ kann die Ausgabesprache gesteuert werden.

Beispiel: `details.php?lang=en&uri=http://leipzigdata.de/Ort/Sommerbad_Suedost`.

search.php: Diese Datei wird nur via Ajax-Request aufgerufen. Über den GET-Parameter „search“ kann ein String übergeben werden, nach dem im RDF-Graphen gesucht wird. Je nachdem, ob ein Ort mit der entsprechenden Bezeichnung gefunden wird, wird eine entsprechende JSON codierte Antwort erzeugt. Über den zweiten Parameter „lang“ kann die Rückgabesprache gesteuert werden.

Beispiel: `search.php?lang=en&uri=Werkll`

group.php: Diese Datei wird nur via Ajax-Request aufgerufen. Über den GET-Parameter „group“ kann eine Kategorie übergeben werden. Es wird im RDF-Graphen nach Orten gesucht, die zu dieser Kategorie gehören. Die gefundenen Orte werden in einer JSON codierten Antwort zurückgegeben. Über den zweiten Parameter „lang“ kann die Rückgabesprache gesteuert werden.

Beispiel: `search.php?lang=en&group=Nachtleben`

[7] Details siehe `http://holdirbootstrap.de/css/`.
[8] Details siehe `http://holdirbootstrap.de/komponenten/#nav/`.

3.6 Beschreibung der einzelnen JS-Dateien

Die Anwendung arbeitet mit zwei eigenen JS-Dateien sowie einer größeren Anzahl von globalen JS-Variablen, die alle in der Datei `custom-map.js` definiert werden. Deshalb *muss* diese Datei zuerst und *nach* der Einbindung von `leaflet.js` geladen werden. Weiterhin *dürfen diese drei Dateien erst am Ende der Seite* eingeladen werden, da sonst die Karte nicht angezeigt wird. Grund: der map-Tag muss im DOM-Baum angelegt sein, bevor er initialisiert wird. Alternative Lösung mit `<body onload="drawmap();">` und Kapselung des Inhalts von `custom-map.js` in einer Funktion `drawmap()`.

custom-map.js: Mit dieser JS-Datei wird die Karte aufgebaut. Initialisiert mit

```
var map = L.map('map').setView([51.3400, 12.3800], 12);
```

die im DOM-Baum als `<div id="map"></div>` eingebundene Karte und bindet daran eine Mapbox-Kartendarstellung über `L.tileLayer(...)`.

`allLocations(...)` liest über einen Ajax-Request die RDF-Quelle als JSON-Datei ein und legt zu jedem jsp:Ort einen Marker an, der sowohl im globalen Array `allmarker` abgelegt als auch über `map.addLayer(...)` zur Kartendarstellung hinzugefügt wird.

helper.js: Definiert die folgenden weiteren JS-Funktionen

- `contentloader(uri, lang)` – lädt über `details.php` den Content eines jsp:Orts in der angegebenen Sprache in den Informations-Container.
- `searchRequest(searchKey)` – wertet über `search.php` eine Suchanfrage aus und baut aus der Rückgabe (Liste aller passenden jsp:Ort-URIs) die Alert-Zeile sowie die Map neu auf.
- `getGroupLocations(group)` – erzeugt über `group.php` eine Liste aller jsp:Ort-URIs der angegebenen Kategorie und baut die Alert-Zeile sowie die Map neu auf.
- `addLocation(...)` – Hilfsfunktion, erzeugt eine neue Location-Information.
- `getUserLanguage()` – Kennung der aktuellen Sprache aus dem div-Element `<div id="languageStore">de</div>` auslesen.
- `removeAllMarkers()` – (lokale) Funktion zum Löschen aller Marker.
- `setFocusToMarker(uri)` – Popup zu einem Marker öffnen.

3.7 Applikations-Einstellungen

Die Applikationseinstellungen sind im Verzeichnis „config“ als PHP-Datei gespeichert. Im Moment sind dort lediglich drei Einträge hinterlegt: Die PHP-Mindest-Version, welche die Applikation benötigt, der Debug-Mode kann aus und angeschaltet werden und die Standardsprache festgelegt werden.

4 Offene Aufgaben

- Das Parsen der RDF-Daten muss weiter verbessert werden und robuster/flexibler gegenüber Fehlern werden. Zum Beispiel muss die Anzeige angepasst werden, wenn keine Öffnungszeiten, Koordinaten oder Kontaktdaten verfügbar sind.

- Weitere Refaktorierung des alten Codes: Einige Funktionalitäten des alten Codes fehlen noch bzw. müssen überarbeitet werden.

  Zum Beispiel die automatisierte Suche nach Koordinaten, wenn im RDF-Graph keine gefunden werden konnten. Oder regelmäßiges Update der lokal zwischengespeicherten RDF-Daten über einen Cronjob.

- JavaScript Erweiterung

  Es sollte evaluiert werden, ob sich aus Performancegründen einige der Aufgaben vom Server auf den Client verlagern und via JavaScript lösen lassen, um eine schnellere Reaktionsfähigkeit der Benutzeroberfläche zu ermöglichen.

- Kategorien/Eigenschaften

  Im aktuellen RDF-Graphen gibt es mehrere Hauptkategorien, denen verschiedene Orte zugeordnet sind. Jede dieser Hauptkategorien besitzt eine ganze Reihe weiterer Subkategorien. Momentan ist es nicht möglich, diese automatisch generiert aus dem RDF-Graph auszulesen. Es sollte eine Möglichkeit gefunden werden, den RDF-Graphen nach den Subkategorien zu durchsuchen, dabei kann sich am schon vorhandenen Code orientiert werden. Im nächsten Schritt muss eine Filtermöglichkeit nach diesen Subkategorien eingebaut werden kann.

5 Frequently Asked Questions

- Es wird keine Karte angezeigt.

  Symptom: Die Marker werden auf der Karte angezeigt, aber es fehlen die Kartendaten selbst.

  Lösung: Wie oben beschrieben, werden die Karten selbst über einen externen Dienstleister bereitgestellt. Aktuell ist dieser Dienstleister *Mapbox*. Sollte die Karte nicht mehr korrekt angezeigt werden, liegt dies vermutlich daran, dass der API-Key abgelaufen ist oder der Anbieter sein Modell umgestellt hat. Ist der Anbieter nicht mehr verfügbar, muss ein neuer Kartenprovider gefunden werden. Hier lohnt es sich einen Blick auf die Quickstart-Seite[9] von Leafletjs.com zu werfen. In der Applikation wird Leaflet.js verwendet, um die Marker auf der Karte anzuzeigen. Oft wir hier auch ein Vorschlag zu einem aktuell verfügbaren Kartenprovider gemacht. Ist ein neuer API-Key verfügbar, muss dieser in der Datei „custom-map.js“ (public/js) auf Zeile 11 eingetragen werden.

- Auf der Hauptseite wird keine Schrift angezeigt.

  Symptom: Auf der Hauptseite „index.php“ wird kein Text angezeigt und das Webseitenlayout ist leicht verschoben.

[9] `http://leafletjs.com/examples/quick-start.html`

Lösung: Möglicherweise liegt die Ursache darin, dass das Lesen von INI-Dateien auf dem Webserver verboten ist. Hierfür muss in der globalen PHP-Konfiguration des Rechners in der *php.ini* die Option *parse_ini_file* aktiviert sein.

- Es fehlen Beschriftungen aus den Übersetzungsdateien.

  Symptom: Es wurde eine neue Übersetzungsdatei zur Unterstützung einer neuen Sprache erzeugt (data/translation/*.ini). Nach Aktivierung der Sprache wird auf der Webseite kein Text mehr angezeigt.

  Lösung: Es muss bei den Übersetzungsdateien darauf geachtet werden, dass die Values immer von Anführungszeichen umschlossen sind. Ansonsten kann es zu Fehlern in der Funktion kommen, die die Übersetzungsdateien parst.

```
1  [website]
2  tabTitle = "Youth City Plan Leipzig"
3  [menu]
4  label = "Youth City Plan Leipzig"
5  map = Map
6  aboutUs = "About us"
7  contact = "Contact"
8  languageButton = "Choose Language"
9  [LocationDescription]
10 placeDescription = "Topography"
11 opening = "Opening Hours "
12 contact = "Contact"
13 properties = "Properties"
14 chooseEntry = "Please choose a location ..."
15 entryNotAvaiable = "This Information is currently unavaiable"
16 [categories]
17 allCategories = "All Locations"
18 noCategorie = "No Categories"
19 [footer]
20 copyright = "Youth City Plan Leipzig"
21 copyrightdate = 2014
```

6 Änderungen

12.02.2015, HGG. Folgende Änderungen wurde vorgenommen:

- Umgang mit der Spracheinstellung: Statt `validateLanguage(language)` wird dies nun lokal, wo dies benötigt wird, durch einen Aufruf von `getLanguage()` realisiert. Diese neue Funktion ist in `translate.php` definiert, womit `helper.php` obsolet ist und gelöscht wurde.

- `buildHTMLPage.php` wurde nach `index.php` verschoben und entsprechend modifiziert, da weitere Webseiten einen ähnlichen Aufbau haben. Damit ist auch die Funktion `pageTemplate()` überflüssig und wurde gelöscht.

- Einheitliche Datenquelle ist das Verzeichnis `data/rdf`. Das war in `custom-map.js` anzupassen. Das zweite Verzeichnis `public/data` wurde gelöscht.

- Das Verzeichnis `data/rdf` enthält weiterhin ein Skript `trans.php`, mit dem eine JSON-RDF-Datei in eine Turtle-Datei verwandelt werden kann. Damit wurde aus dem Original `jugendstadtplan.json` die Turtle-Datei `JSP-13.ttl` erzeugt, dort die fehlende Information für die Kategorie jsp:Bildung nachgetragen und das Ganze in eine neue Version

der RDF-JSON-Datei `jugendstadtplan.json` zurückverwandelt. Es zeigt sich in der Anwendung, dass EasyRDF solche RDF-JSON-Dateien deutlich schneller lädt als die entsprechende Turtle-Datei.