Dokumentation - Jugendstadtplan Leipzig

1. Installation:

1.1 Requirements:

- → Webserver: z.B. Apache, Nginx
- → PHP: mindestens 5.1.X
  - → PHP muss die Rechte besitzen auf JSON und INI Dateien zugreifen zu können
- → Composer (https://getcomposer.org/)

1.2 Installation:

- → Sourcecode Jugendstadtplan Leipzig runterladen (https://github.com/unlimitedone/Karten/tree/master/Jugendstadtplan/Plath)
- → Daten auf Webspace entpacken
- → Composer ausführen (lädt benötigte Bibliotheken) *„php composer.phar install“*
- → im Webbrowser die Datei *„index.php“* im Ordner *„public“* aufrufen

2. Applikation

2.1 Ordnerstruktur Applikation:

- → config (Ordner)
- → data (Ordner)
- → module (Ordner)
- → public (Ordner)
- → vendor (autogeneriert durch Composer)
- → composer.json

2.2 Aufbau der Applikation:

„config“: Dieser Ordner enthält alle Einstellungen der Applikation.
„data“: In diesem Ordner werden alle Daten der Applikation abgelegt. Zum Beispiel die lokalen RDF-Daten oder die Übersetzungsdateien.
„module“: Dieser Ordner enthält die Funktionalitäten der Software. Alle Funktionalitäten sind in Form von PHP-Funktionen in diesem Ordner abgelegt.
„public“: Dieser Ordner stellt die Webschnittstelle dar. Es ist der einzige Ordner der von außen erreichbar sein muss. Alle anderen Ordner können z.B. durch eine htaccess-Datei geschützt werden.
„vendor“: Dieser Ordner wird automatisiert durch Composer erstellt. Er enthält alle importierten Bibliotheken bzw. Applikationsabhängigkeiten.
„composer.json“: Diese Datei dient zur Steuerung von Composer und enthält alle Abhängigkeiten.

2.2.1 Benutze Bibliotheken:

→ RDF Framework – EasyRDF für PHP (http://www.easyrdf.org/) integriert und verwaltet über Composer
→ Web Framework - Bootstrap 3 (http://getbootstrap.com/)
→ Map Framework – Leaflet (http://leafletjs.com/)
→ HTML/CSS Flaggen – (http://lipis.github.io/flag-icon-css/)

2.2.2 Benutzte externe Dienste:

→ Kartenprovider – Mapbox (https://www.mapbox.com/)
Die auf dem Jugendstadtplan angezeigte Karte muss entweder selbst gehostet werden oder über einen Dienstleister bereitgestellt werden. Momentan werden die Karten über den Dienstleister Mapbox bezogen. Die Interaktion mit den von Mapbox gelieferten Karten, wird über die JavaScript-Bibliothek Leaflet ermöglicht.

2.2.3 Webschnittstelle:

Die Webschnittstelle ist möglichst leichtgewichtig gestaltet, da die Kernfunktionalitäten im Module Ordner implementiert sind. Die Hauptaufgabe ist, dass entgegennehmen der Requests der Nutzer und der Aufruf der Entsprechenden Kernfunktionalität.

„index.php“: Diese Datei hat die Aufgabe die komplette Website auszuliefern. Über den GET-Parameter „lang“, kann die Website in einer bestimmten Sprache angefordert werden. 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 Ordner „config“ angepasst werden.
Beispiel: „index.php?lang=de“ oder „index.php?lang=en“

„details.php“: Diese Datei wird nur via Ajax-Requests aufgerufen. Über den GET-Parameter „uri“ kann ein Ort angefragt werden. Wobei der Ort selbst eine URI sein muss. Wird der entsprechende Ort im RDF Graphen gefunden, werden entsprechende Details zurückgegeben. Über den zweiten Parameter „lang“ kann die Rückgabesprache gesteuert werden.
Beispiel: details.php?lang=en&uri=http://leipzigdata.de/Ort/Sommerbad_Suedost

„search.php“: Diese Datei wird nur via Ajax-Requests 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 werden konnte, 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-Requests 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

2.2.4 Multilingualität:

Die Applikation ist so entwerfen, dass verschiedene Sprachen unterstützt werden und weitere Sprachen ohne Programmierkenntnisse hinzugefügt werden können. Aller zu diesem Zeitpunkt verfügbaren Sprachen liegen im Verzeichnis „data/translation“. Alle verfügbaren Übersetzungen liegen als INI-Dateien vor, was ein leichtes editieren und lesen der Dateien ermöglicht. Um eine weitere Sprache hinzuzufügen, muss lediglich eine der vorhandenen Sprachen in die gewünschte Sprache übersetzt werden und in selbigen Verzeichnis abgelegt werden.

2.2.5 Applikations-Einstellungen:

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

3. Zukünftige Aufgaben:

3.1 RDF-Daten-Parsen:

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

3.2 Weitere Refrakturierung des alten Codes

Einige Funktionalitäten des alten Codes fehlen noch bzw. müssen überarbeitet werden. Zum Beispiel automatisierte suche nach Koordinaten wenn im RDF-Graph keine gefunden werden konnten. Oder regelmäßig update der lokal gecachten RDF Daten, möglichweise über einen Cronjob.

3.3 JavaScript Erweiterung

Es sollte evaluiert werden, ob man aus Performance gründen eventuell einige der Aufgaben vom Server auf den Client überträgt und via JavaScript löst, um eine schnellere Reaktionsfähigkeit der Benutzeroberfläche zu ermöglichen.

3.4 RDF Kategorien/Eigenschaften

Im Momentanen RDF-Graph gibt es mehrere Hauptkategorien, denen verschiedene Orte zugeordnet sind. Jeder dieser Hauptkategorien besitzt eine ganze Reihe weiterer Subkategorien. Momentan ist es nicht möglich, diese automatisch generiert aus dem RDF-Graph auszulesen. Es sollte nach einer Möglichkeit gefunden werden den RDF-Graphen nach den Subkategorien zu durchsuchen, dabei kann sich an den schon vorhandenen Code orientiert werden. In nächsten Schritt muss dann realisiert werden, dass über die Webseite nach die Subkategorien gefiltert werden kann.