


EINE WEBBASIERTE GIS-ANWENDUNG ZUR LOKALISIERUNG VON SEHENSWÜRDIGKEITEN IM STADTKERN DER STADT KÖTHEN (ANHALT)

Eine Dokumentation
der Projektarbeit im Modul
Geodatenbanken

23.02.2020

 **Alexander Prinz**
Matr.-Nr.: 4069949
Studiengang: Data Science (M. Sc.)
Fachbereich: Informatik und Sprachen


Hochschule Anhalt

Anhalt University of Applied Sciences

Eigenständigkeitserklärung

Hiermit bestätige ich, Alexander Prinz, geboren am 26.02.1986, die folgende Arbeit selbstständig verfasst und nur die mir gesetzlich zustehenden Mittel zum Verfassen wissenschaftlicher Arbeiten genutzt zu haben. Genutzte Software anderer Autoren, welche mir beim Bearbeiten dieses Projektes dienten, habe ich gemäß den Prinzipien des wissenschaftlichen Arbeitens und dem Urheberrecht entsprechend kenntlich gemacht.

Eingereicht am: 23.02.2020

A handwritten signature in blue ink, consisting of a stylized 'P' followed by a horizontal line with a small upward curve at the end.

Danksagung

Ich möchte mich an dieser Stelle ganz herzlich bei Herrn Prof. DR.-ING. HOLGER BAUMANN als Dozent des Moduls „Geodatenbanken“ für das von ihm vermittelte Wissen sowie die Möglichkeiten dieses Projektes bedanken.

Zusammenfassung

Ziel des Projektes war die Entwicklung eines webbasierten GIS zur Lokalisierung von Sehenswürdigkeiten im Stadtkern der Stadt Köthen (Anhalt). Die Anwendung ermöglicht es dem Nutzer, sowohl die bereits in das System eingepflegten Sehenswürdigkeiten zu finden und Daten zu diesen Sehenswürdigkeiten abzurufen, als auch durch eine Formularübermittlung eigene Sehenswürdigkeiten und deren Daten in das System zu integrieren.

Die Sehenswürdigkeiten sind auf einer Karte als Icons an den entsprechenden Stellen markiert. Diese Icons repräsentieren Hyperlinks auf eine Seite, auf der Bild- und Textinformation zu der Sehenswürdigkeit bereitgestellt sind.

Die Daten werden in einer SQL-Datenbank verwaltet und durch diverse Programme abgerufen oder verändert.

Abbildungsverzeichnis

Abbildung I:	Karte des Stadtkerns von Köthen.....	3
Abbildung II:	HTML-Quellcodeausschnitt zur Kalibrierung der Koordinaten- transformation	5
Abbildung III:	Schema der Datenbank.....	7
Abbildung IV:	SQL-Skript zur Erstellung der Datenbank.....	8
Abbildung V:	Quellcode der Methode <code>is_to_near_or_equal_or_not_in_area()</code>	10
Abbildung VI:	Exemplarischer HTML-Quellcode der Seite einer Sehenswürdigkeit.....	12
Abbildung VII:	HTML-Quellcodeabschnitt des Eingabeformulars der Hauptseite.....	13
Abbildung VIII:	Exemplarische Datei bzgl. der Datenübergabe von <i>file_upload.php</i> nach <code>do_files_into_database.py</code>	13
Abbildung IX:	Eingabeformular in gerenderter Form.....	14
Abbildung X:	Karte des Stadtkerns von Köthen in vergrößerter Form.....	i
Abbildung XI:	Visuelles Schema der Seitenverlinkung	ii

Tabellenverzeichnis

Tabelle 1:	Die einzelnen SHP-Dateien, welche zur Erstellung der Karte aus externen Quellen genutzt wurden.....	2
------------	--	---

Formelzeichen und Definitionen

D	Definitionsbereich D
$d(v_1, v_2)$	Abbildung d der Vektoren v_1 und v_2 auf D als Distanz beider Vektoren zueinander
f	Abbildung f
λ	Skalar Lambda zur Skalierung der zu transformierenden Abszisse
μ	Skalar Mü zur Skalierung der zu transformierenden Ordinate
\mathbb{N}^2	Menge aller Vektoren der Dimension 2 deren Elemente natürliche Zahlen sind
\mathbb{N}	Menge aller natürlichen Zahlen
\mathbb{R}^2	Menge aller Vektoren der Dimension 2 deren Elemente reelle Zahlen sind
\mathbb{R}	Menge der reellen Zahlen
v_1	Vektor v_1
v_2	Vektor v_2
V	Vektorraum V
W	Vektorraum W
\vec{x}_{UTM}	Koordinate aus dem UTM-Raum
\vec{x}_{UTM_ref}	Referenzkoordinate aus dem UTM-Raum
\vec{x}_{px}	Koordinate aus dem Pixel-Raum
\vec{x}_{px_ref}	Referenzkoordinate aus dem Pixel-Raum
x_{UTM}	Abszisse der Koordinate aus dem UTM-Raum
y_{UTM}	Ordinate der Koordinate aus dem UTM-Raum
x_{px}	Abszisse der Koordinate aus dem Pixel-Raum
y_{px}	Ordinate der Koordinate aus dem Pixel-Raum
$\ \cdot \ $	euklidische Norm
$\{ \cdot \}$	Menge
$\lceil \cdot \rceil$	Aufrunden-Funktion zur Abbildung einer reellen Zahl auf eine natürliche Zahl
$\lfloor \cdot \rfloor$	Abrunden-Funktion zur Abbildung einer reellen Zahl auf eine natürliche Zahl
$\begin{pmatrix} \lambda & 0 \\ 0 & -\mu \end{pmatrix}$	2×2 Tensor über 4 Tensorelemente aus $\{0, \lambda, -\mu\}$

Inhalt

1. Konzept/Lastenheft	1
2. Die Entwicklungsumgebung	2
3. Erstellung der Karte des Stadtkerns von Köthen	2
4. GPS-Pixel-Koordinatentransformation.....	4
5. Die Datenbank	7
6. Datenbankenabfragen und Operationen mit den Daten.....	9
7. Der Programmablauf zur Erzeugung neuer Sehenswürdigkeiten	11
Quellen-/Autoren-/Entwicklerverzeichnis	15
Anhang.....	i
Anhang 1	i
Anhang 2	ii

1. Konzept/Lastenheft

Eine webbasierte GIS-Anwendung

zur Lokalisierung von Sehenswürdigkeiten im Stadtkern der Stadt Köthen (Anhalt)

Entwickler: Alexander Prinz

Zielsetzung: Eine webbasierte GIS-Anwendung zur Lokalisierung von ausgewählten Sehenswürdigkeiten mit zugeordneten Bildern und textlichen Informationen im Stadtkern der Stadt Köthen (Anhalt)

Grund: Projektarbeit/Prüfungsleistung im Modul Geodatenbanken

Die Anwendung soll eine einfache Weboberfläche bieten, auf der die Karte des Stadtkerns von Köthen zu sehen ist. Die Karte soll dabei alle grundständigen Objekte abbilden. Größere Grünflächen (Parkanlagen/Friedhöfe) sollen als Polygonzüge zu sehen sein. Urbane Objekte, wie Gebäude und Straßen, sollen als Polygonzüge bzw. Linienzüge enthalten sein. Wichtige bzw. bedeutsame Straßen erhalten Straßennamen. Des Weiteren sollen bestimmte Objekte mit dem (willkürlichen) Status einer Sehenswürdigkeit durch ein entsprechendes Icon in der Karte sichtbar gemacht werden. Beim Klicken dieses Icons soll die Sehenswürdigkeit über eine eigenständige Seite in Form einer dort platzierten Bilddatei sichtbar gemacht werden. Zudem sollen dort Textinformationen sichtbar gemacht werden. Über ein Upload-Formular soll dem Nutzer das Hinzufügen weiterer Sehenswürdigkeiten ermöglicht werden.

Die Karte wird mit Hilfe einer GIS-Software erstellt. Features, wie urbane Objekte (Flurstücke, Straßen, Gebäude) sowie Gewässer und die nötige Beschriftung jener Features, werden statisch in die Karte integriert. Die Karte liegt als unveränderliche Version auf dem Webserver in Form einer Pixel-Grafik. Per Datenbank sind dynamische Features, genauer die Sehenswürdigkeiten, abrufbar und werden beim Aufruf der Seite als Icons auf der Karte gerendert. Als Datenbank wird eine SQL-Datenbank dienen. Die Interaktion zwischen Nutzer, Webserver und Datenbank wird mittels verschiedener Skriptsprachen und damit realisierter Programme ermöglicht.

2. Die Entwicklungsumgebung

Die Entwicklungsumgebung besteht aus den im Softwarepaket XAMPP (SEIDLER ET AL., 2019) enthaltenen Komponenten MARIADB (MARIADB FOUNDATION, 2009) als SQL-Datenbank, dem PHP-Interpreter (LERDORF ET AL. 1995) und dem Apache HTTP Server (APACHE SOFTWARE FOUNDATION, 1995). Außerhalb des Pakets wurde der PYTHON-Interpreter (VAN ROSSUM, 1991) genutzt. Die Interaktion zwischen Webserver, PHP-Interpreter und SQL-Datenbank ist in XAMPP optimal auf Entwicklung ausgelegt. Die Datenbank MARIADB ist ein Fork des Datenbankverwaltungssystems MYSQL von SUN MICROSYSTEMS ET AL., 1995, welches heute jedoch der ORACLE CORPORATION gehört. Aus diesem Grund entschieden die Entwickler von XAMPP auf den MYSQL-Fork MARIADB zu wechseln, anstatt weiterhin MYSQL im Paketbestand zu halten. Syntaktisch gibt es jedoch keinen Unterschied zwischen beiden Systemen. Die Anbindung von PYTHON an die Datenbank passiert mit jenen Softwarepaketen, welche eigentlich zur Kommunikation mit MySQL-Datenbanken entwickelt wurden.

Um Quellcode zu schreiben, wurden als Editoren sowohl NOTEPAD++ (HO, 2003) sowie VISUAL STUDIO CODE (MICROSOFT, 2015) genutzt.

3. Erstellung der Karte des Stadtkerns von Köthen

Die Karte wurde mit Hilfe der Software QGIS (QGIS DEVELOPMENT TEAM, 2020) erstellt. Sie zeigt den Stadtkern in einem Ausschnitt von etwa 1,7 km in östliche Richtung × 1,2 km in nördliche Richtung verteilt um den Marktplatz (siehe Abbildung I).

Die erforderlichen SHAPE-Dateien zur Visualisierung von Gebäudegrundflächen, Grünanlagen (Friedhöfe und Parks), Straßennetz, Parkplatzflächen sowie Kirchen wurden von GEOFABRIK (GEOFABRIK, 2007) frei heruntergeladen. Die folgende Tabelle zeigt die Zuordnung der Objekte zu dem Dateinamen der genutzten SHAPE-Daten von GEOFABRIK.

Objekt	Dateiname
Parkplätze	<i>gis_osm_traffic_a_free_1.shp</i>
Gebäude	<i>gis_osm_buildings_a_free_1.shp</i>
Kirchen	<i>gis_osm_pofw_a_free_1.shp</i>
Friedhöfe und Parks	<i>gis_osm_pois_a_free_1.shp</i>
Straßen	<i>gis_osm_roads_free_1.shp</i>
Gewässer	<i>gis_osm_water_a_free_1.shp</i>

Tab.: 1 Die einzelnen SHP-Dateien, welche zur Erstellung der Karte aus externen Quellen genutzt wurden.

Lediglich die Straßennamen mussten eigenständig eingepflegt werden. Dazu wurde eine neue SHAPE-Datei mit dem Namen *Straßen* im Projekt angelegt, welche auf Basis von Linienzügen basiert. Für jede Straße wurde eine Linie in die Datei durch das „Linienobjekt hinzufügen“ Werkzeug aufgenommen und eine individuelle Identifikationsnummer vergeben. In der Attributtabelle der SHAPE-Datei wurde dann eine weitere Spalte neben der Identifikationsnummer erzeugt und der Straßename zu der entsprechenden Identifikationsnummer hinzugefügt. Durch die Beschriftungsfunktion wurden die Straßennamen dann in die Karte eingefügt. Die Parkplatzflächen wurden mit einem entsprechenden Piktogramm, welches normgerecht für die Kennzeichnung von Parkplatzflächen genutzt wird, in der Karte kenntlich gemacht. Die Straßennamen wurde via OPENSTREETMAP (OPENSTREETMAP FOUNDATION, 2004) recherchiert. Auch die Gewässernamen wurde auf diese Art eingepflegt, sofern sie in OPENSTREETMAP zu finden waren. Die zusätzlichen Daten und ihre Projektzuordnungen, wie ID und Attributnamen, wurden in der Date *Objekte.dat* festgehalten. Zudem wurde im linken unteren Kartenabschnitt eine Legende mit den Objekten: Parkplätzen, Straßen, Gewässer, Gebäude sowie Friedhöfe und Parks erzeugt und zusätzlich mit einem Nordpfeil versehen. Über die Legende wurde ein Textfeld erzeugt, welches den Titel „Stadtkern von Köthen“ enthält. Rechts neben der Legende wurde außerdem ein Maßstab zugefügt. Am rechten unteren Kartenabschnitt wurde als weiteres Textfeld eine Information abgebildet, dass die Karte auf Basis von Daten von den bereits genannten Quellen im Jahr 2020 erstellt wurde.



Abb. I: Karte des Stadtkerns von Köthen

Die Karte repräsentiert einen in östliche Richtung 1,7 km × 1,2 km in nördliche Richtung messenden Ausschnitt der Stadt Köthen rund um den Marktplatz. Sie enthält Objekte wie Parkplätze, Straßen und deren Namen, Gewässer und der Namen (falls bekannt), Gebäude, sowie Friedhöfe und Parks. Letzte beiden bilden eine Einheit. Es wurde im unteren linken Kartenabschnitt eine Legende hinzugefügt, welche die genannten Objekte zuordnet. Des Weiteren, wurde rechts neben der Legende ein Maßstab sowie ein Nordpfeil platziert. Zudem ist der Titel der Karte über der Legende zusehen. Im rechten unteren Abschnitt der Karte befindet sich ein Hinweis, dass die Karte mit Daten von GEOFABRIK und OPENSTREETMAP im Jahr 2020 von mir erstellt wurde. Zudem wurde meine aktuelle Hochschulmailadresse dort hinzugefügt. Die Karte ist in einer größeren Version im Anhang 1 zu finden.

4. GPS-Pixel-Koordinatentransformation

Die Sehenswürdigkeiten werden auf der Karte als Hyperlinks repräsentiert, welche durch Punkte auf der Karte dargestellt werden und durch klicken auf diese Punkte bzw. Icons auf eine eigenständige Seite weisen. Die sich dann öffnende Seite zeigt Bilder sowie Textinformationen zu der Sehenswürdigkeit.

Die Sehenswürdigkeiten und ihre Daten, insbesondere der Raumbezug durch die GPS-Koordinaten, werden aus der SQL-Datenbank abgerufen. Damit aber der Link bzw. dessen Icon in der Karte an der richtigen Stelle platziert wird, müssen die GPS-Koordinaten in Pixel-Koordinaten, passend zu der Karte, transformiert werden. Dies passiert über zwei Transformationen: Die erste Transformation bildet die GPS-Koordinaten auf UTM (WGS84, Z32, N) Koordinaten ab. Dazu wird eine Python-Bibliothek namens *utm*¹ und deren Funktion *fromlatlon()* genutzt. Diese Funktion erhält als Parameter die Latitude sowie die Longitude und gibt die UTM-Koordinaten zurück. Die Validität der Umrechnung wurde durch die Seite www.koordinaten-umrechner.de durch einige Testumrechnungen verifiziert.

Um von den UTM-Koordinaten auf die relativen Pixel-Koordinaten der Karten abzubilden, wurde eine eigene Funktion bzw. Methode entwickelt. Dabei wird davon ausgegangen, dass die Transformation von UTM-Koordinaten auf Pixel-Koordinaten homomorpher Natur ist. Es soll dann Folgendes gelten:

$$f: V \rightarrow W, \mathbb{R}^2 \mapsto \mathbb{N}^2, \vec{x}_{UTM} \in V, \vec{x}_{px} \in W \quad (1)$$

Dabei ist f die lineare Abbildung von V nach W , wobei V der Vektorraum über alle UTM-Koordinaten und W der Vektorraum über alle Pixel-Koordinaten ist. Der Vektor \vec{x}_{UTM} enthält die Abszisse x_{UTM} und die Ordinate y_{UTM} . Der Vektor \vec{x}_{px} repräsentiert die korrespondierende Pixelkoordinate bestehend aus x_{px} und y_{px} . Da die Transformation lediglich eine Streckung des Koordinatensystems in x- wie auch in y-Richtung ist, genügt als Transformation die Matrixmultiplikation eines Tensors. Dieser beinhaltet die beiden zu den Achsen korrespondierenden Skalare λ korrespondierend zu x sowie μ korrespondierend zu y und wird mit dem Vektor \vec{x}_{UTM} verschoben um einen Referenzpunkt/Vektor \vec{x}_{UTM_ref} matrixmultipliziert.

¹ Zu dem Paket *utm* existieren keine Informationen über den Entwickler oder jedwede andere zitierbare Informationen. Es wurde über den PYTHON-Standardpaketmanager PIP runtergeladen.

Das Matrixprodukt wird dann mit einem Referenzvektor aus dem Pixel-Koordinatenraum \vec{x}_{px_ref} (siehe Gl. 1) addiert.

Es sei noch zu beachten, dass das Koordinatensystem bzgl. der Ordinaten-Achse bei dem Pixelkoordinatensystem relative zum UTM-Koordinatensystem invertiert ist. Um dies durch Transformation zu realisieren, erhält der Skalar μ im Tensor ein negatives Vorzeichen. Damit der Funktionswert, welcher per se eine rationale Zahl sein kann und somit nicht als Pixelkoordinaten dienen kann, zu einer natürlichen Zahl überführt wird, muss noch auf die erste Nachkommastelle gerundet werden.

Die Abbildungsvorschrift sei somit:

$$f(\vec{x}_{UTM}) := \begin{cases} \left[\begin{pmatrix} \lambda & 0 \\ 0 & -\mu \end{pmatrix} \circ \begin{pmatrix} x_{UTM} - x_{UTM_{ref}} \\ y_{UTM} - y_{UTM_{ref}} \end{pmatrix} + \begin{pmatrix} x_{px_{ref}} \\ y_{px_{ref}} \end{pmatrix} \right], & \text{wenn erste Nachkommastelle} \geq 5 \\ \left[\begin{pmatrix} \lambda & 0 \\ 0 & -\mu \end{pmatrix} \circ \begin{pmatrix} x_{UTM} - x_{UTM_{ref}} \\ y_{UTM} - y_{UTM_{ref}} \end{pmatrix} + \begin{pmatrix} x_{px_{ref}} \\ y_{px_{ref}} \end{pmatrix} \right], & \text{sonst} \end{cases} \quad (2)$$

Es wurden 3 Punkte in QGIS mit homogener Verteilung auf der Karte gewählt und deren UTM-Koordinaten notiert. Zu jedem dieser Punkte wurde die korrespondierende Pixelkoordinate durch manuelles Anpassen gefunden. Dazu wurde ein HTML-Code erzeugt, welcher die Karte als Webseite abbildet und den schon erwähnten Hyperlink durch ein räumlich zugeordnetes Icon ermöglicht. Das Icon ist also an der Stelle in der Karte platziert, wo sich die entsprechende Sehenswürdigkeit befindet. Das Icon bzw. der Hyperlink wird bzgl. seiner Platzierung durch diese Pixelkoordinaten definiert. Die Koordinate wurde solange manuell verändert, bis sich das Icon auf der Karte an der richtigen Stelle befand. Die Pixelkoordinaten wurden dann ebenfalls notiert. Die Datei, in der die UTM- und zugehörigen Pixelkoordinaten aufgenommen wurden, ist *coord_transf.dat*.

In Abbildung II ist ein Quellcodeausschnitt des HTML-Dokuments zu sehen.

```
001 <div>
002 <figure id="map">
003     <svg version="1.1" xmlns="http://www.w3.org/2000/svg...
004     <image width="351" height="248" xlink:href="karte...
005     <a xlink:href="test.png"><circle cx="79" cy="56" r="1.8"...
006     </svg>
007 </figure>
008 </div>
```

Abb. II: HTML-Quellcodeausschnitt zur Kalibrierung der Koordinatentransformation

Die Abbildung zeigt einen wenig Zeilen umfassenden Abschnitt aus dem HTML-Quellcode, welcher benutzt wurde, um die Transformationsparametrisierung zu kalibrieren. Einige, für diese Darstellung nicht relevante Code-Zeilen, wurden verkürzt (durch drei aufeinander folgende Punkte gekennzeichnet). Die beiden Parameter cx und cy in Zeile 5 repräsentieren die Pixelkoordinate. Diese Werte wurden manuell durch Probieren solange verändert, bis das Icon mit dem Hyperlink auf der richtigen Position lag.

Um λ und μ zu schätzen, wurden über diese Daten der mittlere λ - sowie μ -Wert geschätzt. Gleichungen 3 und 4 zeigen den mathematischen Zusammenhang. Das Prinzip ist, dass alle Strecken zwischen zwei Punkten aus V das gleiche Verhältnis in W haben müssen. Also ist:

$$\lambda(x_{UTM_1} - x_{UTM_0}) = x_{px_1} - x_{px_0} \Leftrightarrow \lambda = \frac{x_{px_1} - x_{px_0}}{x_{px_1} - x_{px_0}} \quad (3)$$

$$\mu(y_{UTM_1} - y_{UTM_0}) = y_{px_1} - y_{px_0} \Leftrightarrow \mu = \frac{y_{px_1} - y_{px_0}}{y_{px_1} - y_{px_0}} \quad (4)$$

Bei N Daten würden so über $N^2 - N$ mögliche Punktbeziehungen bzw. Abstände die λ - und μ -Werte berechnet und arithmetisch gemittelt. Die Funktion `get_lamda_mu_and_opt_ref_point()` aus der Projektbibliothek wurde dazu entwickelt und genutzt. Einzelheiten bzgl. des Ablaufs kann aus dem Quellcode bzw. dessen Kommentaren entnommen werden. Aus linearalgebraischer Sicht, würden zwei Punkte für die Kalibrierung natürlich ausreichen. Da die Daten durch die manuelle Aufnahme der Punkte jedoch fehlerbehaftet sein können, ist es daher sinnvoll, den Fehler durch die Mittelung/Kalibrierung von λ und μ zu minimieren. Zudem wurden auch die optimalen Referenzpunkte $(x_{UTM_{ref}}, y_{UTM_{ref}})^T$ sowie $(x_{px_{ref}}, y_{px_{ref}})^T$ durch Fehlerminimierung, jedoch in dem Fall auf Basis der kleinsten quadratischen Fehler, geschätzt.

Um diese Dokumentation nicht zu sehr in eine allzu mathematische Richtung zu lenken, möchte ich auf die Mathematik der letzten Methode nicht genauer eingehen. Zum genauen Verständnis der Numerik und Algorithmik hilft der entsprechende Programmcode der Funktion `get_lamda_mu_and_opt_ref_point()`.

5. Die Datenbank

Die Datenbank basiert, wie bereits in Kapitel (Die Entwicklungsumgebung) erwähnt, auf MARIADB welche dem Softwarepaket XAMPP beiliegt.

Sie soll die Sehenswürdigkeiten hinsichtlich ihrer GPS-Koordinaten, Namen und textlichen Informationen sowie Bilddateinamen speichern. Die Bilder werden nicht direkt in der Datenbank gespeichert, sondern in einem gesonderten Verzeichnis. Jedoch wird der Dateiname der Bilder in der Datenbank gespeichert, sodass bei der beim Seitenabruf erfolgenden Datenbankenabfrage die Pfade zu den Bilddateien generiert werden können. Die Datenbank wurde mit dem Namen *koethen* via einem SQL-Skript angelegt (siehe Abbildung IV). Sie wurde dem Nutzer *sql_api* mit allen Rechten nutzbar gemacht. Die Datenbank besteht nur aus einer Tabelle, welche den Namen *items* trägt. Die Tabelle besteht aus 5 Spalten, welche die einzelnen Attribute (*name*, *lat*, *lon*, *image_names*, *text_info*) enthält.

In Abbildung III ist das Schema der Datenbank *koethen* zu sehen. Es ist aufgrund nur einer zugehörigen Tabelle nicht als wirkliches Entitäts-Beziehungsmodell zu verstehen, soll aber die Tabelle und ihre Attribute nochmals grafisch darstellen. Die Namen wurden hierzu durch deutsche Begriffe deklariert.

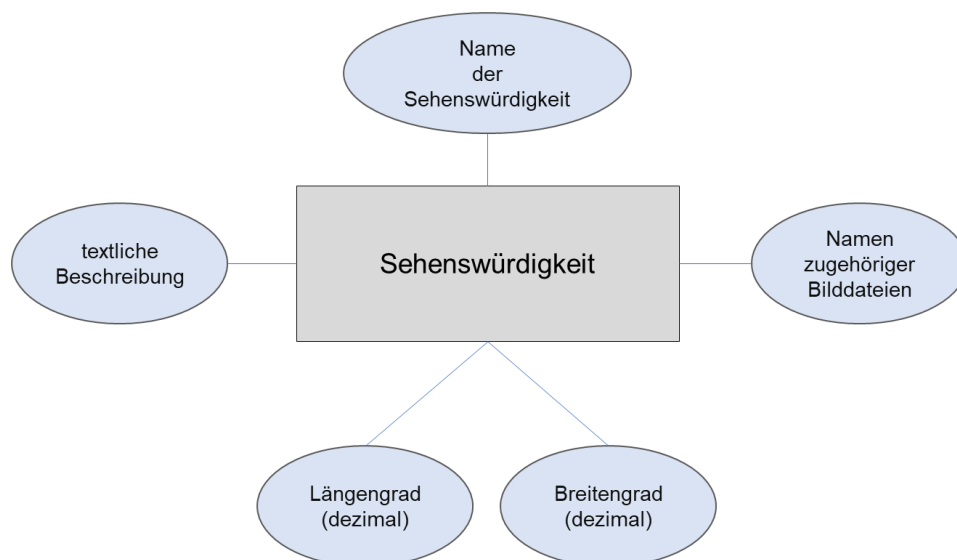


Abb. III: Schema der Datenbank

Die Abbildung zeigt das Schema der Datenbank. Die einzelnen Sehenswürdigkeiten definieren sich durch die Attribute Name der Sehenswürdigkeit, textliche Beschreibung, jeweils Längen- sowie Breitengrad und den Namen möglicher Bilddateien, welche der Sehenswürdigkeit zugeordnet werden können. Da es keine Beziehungen zwischen anderen Entitäten gibt, ist keines der Attribute ein Schlüssel.

```

001 -- erstelle Nutzer py_script und weise ihm das Passwort db_koet_ zu
002 create user 'sql_api'@'localhost' identified by 'db_koet_';
003
004 -- setze die Rechte durch
005 flush privileges;
006
007 -- erstelle eine Datenbank namens koethen
008 create database koethen;
009
010 -- weise dem user py_script die Datenbank koethen mit allen Rechten zu
011 -- die Rechte sollten so natürlich nur in der Entwicklungsphase gelten
012 grant all privileges on koethen . * to 'sql_api'@'localhost';
013
014 -- setze die Rechte durch
015 flush privileges;
016
017 -- gehe in Datenbank koethen
018 use koethen
019
020 -- erstelle die Tabelle items mit allen Attributen
021 create table items(
022 name varchar (50),
023 lat float(32),
024 lon float(32),
025 img_names text,
026 text_info text
027 );

```

Abb. 4: SQL-Skript zur Erstellung der Datenbank

Das Skript erstellt in Zeile 2 einen Nutzer namens *sql_api*, der durch das Passwort *db_koeth_* auf den Standardentwicklungsserver *localhost* Zugang hat. In Zeile 12 werden dem Nutzer alle Rechte zugewiesen (welche natürlich in dieser Form nur im Entwicklungsbetrieb zuerkannt werden sollten). Zeile 15 setzt die zugewiesenen Rechte ohne Serverneustart durch. In Zeile 21 bis 27 wird die Tabelle *items* erstellt, welche die Sehenswürdigkeiten durch die Attribute *name*, *lat*, *lon*, *img_names* und *text_info* repräsentiert. Alle Textzeilen mit einem voranstehenden — sind textliche Information zu dem Skript, welche nicht ausgeführt werden.

6. Datenbankenabfragen und Operationen mit den Daten

Da mir als Datenbankverwaltungssystem MARIADB zur Verfügung stand, konnten die notwendigen Operationen nicht direkt mit den im Datenbankensystem implementierten Funktionen getätigt werden, sondern wurden außerhalb der Datenbank erledigt. Zudem gibt es in dem Projekt keine Beziehungen zwischen einzelnen Entitäten, da es wie bereits in Kapitel 5 erwähnt, nur eine Tabelle gibt. Die Tabelle trägt den Namen *items* und verwaltet pro Sehenswürdigkeit, alle Attribute. Die einzigen Daten, mit denen numerisch operiert wird, sind die GPS-Koordinaten der einzelnen Sehenswürdigkeiten. Sie sind als die beiden Attribute *lat* (Breitengrad) und *lon* (Längengrad) je Sehenswürdigkeit gespeichert. Um außerhalb der Datenbank mit den Daten zu operieren, wurde von mir in PYTHON die Klasse *PY_SQL_API* entwickelt, welche alle nötigen Methoden zum Operieren mit den Daten enthält. Die Klasse ist in der Bibliotheksdatei *PY_lib.py* zu finden.

Die hauptsächliche räumlich-numerische Operation ist neben der in Kapitel 4 vorgestellten Transformation der GPS-Koordinaten in Pixel-Koordinaten die Berechnung der Distanz zweier Koordinaten, etwa um zu prüfen, ob sich die Sehenswürdigkeit im validen Kartenbereich befindet oder zu nah an einer bereits existierenden Sehenswürdigkeit liegt. Die Distanz zweier Punkte lässt sich über die euklidische Norm des Richtungsvektors beider Punkte berechnen. (siehe Gleichung 5 und 6).

Sei: $v_1, v_2 \in V, d: V \rightarrow D, \mathbb{R}^2 \mapsto \mathbb{R}$, wobei v_1 und v_2 die beiden Koordinaten sind und d die Distanz zwischen den beiden Koordinaten als Abbildung dieser Koordinaten aus dem Vektorraum V auf eine reelle Zahl aus D , dann ist die Abbildungsvorschrift repräsentiert durch die euklidische Norm des Richtungsvektors beider Koordinaten. Der Richtungsvektor entspricht der Differenz beider Koordinaten:

$$\|v_2 - v_1\| = \sqrt{(v_1^1 - v_2^1)^2 + (v_1^2 - v_2^2)^2} \quad (5)$$

$$d(v_1, v_2) := \|v_2 - v_1\| \quad (6)$$

Anmerkung: Die hochgestellte Zahl über dem v ist ein Index, welcher auf die Vektorkomponente verweist. Die tiefgestellte Zahl dagegen ist der Index, welcher auf den Vektor selbst verweist.

Die Methode `is_to_near_or_equal_or_not_in_area()` der Klasse `PY_SQL_API` prüft durch diese mathematische Vorschrift, ob die Koordinate einer neu hinzuzufügenden Sehenswürdigkeit zu weit von der Kartenmitte oder zu nah an einem schon existierenden Objekt ist. Abbildung V zeigt einen Quellcode-Ausschnitt aus der Klasse, welche die Methode definiert. Da, wie bereits erwähnt, nicht in der Datenbank operiert wird, müssen die Daten zunächst aus ihr extrahiert werden. Die ebenfalls der Klasse angehörige Methode `get_table_from_db_as_list()` extrahiert die Daten aus der Tabelle und schreibt sie in eine Liste (siehe Abbildung V, Zeile 21). Über diese Liste wird dann mit einer For-Schleife iteriert. In jedem Iterationsschritt werden die Koordinaten der Sehenswürdigkeit der aktuellen Zeile aus der Tabelle in ein NUMPY-Array geschrieben (siehe dazu: OLIPHANT, 2006). In Abbildung V, Zeile 33 wird dann mithilfe der NUMPY-Methode `norm()` aus der `linalg`-Klasse die Distanz zwischen der zu prüfenden Koordinate mit der Iterationsaktuellen Koordinate berechnet. Sofern die errechnete Distanz kleiner oder gleich dem willkürlich gewählten Wert von 0.00008 Koordinatenlängen ist, wird die Schleife abgebrochen und ein Fehlerwert zurückgegeben. Zudem wird in jeder Iteration geprüft, ob der Name der Iterationsaktuellen Sehenswürdigkeit gleich dem der zu prüfenden Sehenswürdigkeit ist.

```

001 def is_to_near_or_equal_or_not_in_area(self, lat, lon, name):
002
003     gps = np.array([lat, lon])
004
005     # frage ab ob Koordinate im nicht erlaubeten Kartenbereich ist.
006     # Wenn ja gib entsprechenden Fehler zurück.
007     if np.linalg.norm(gps - np.array([51.751371, 11.973681])) > 0.005347:
008         return 'err_not_in_area'
009
010
011     table = self.get_table_from_db_as_list()
012     for row in table:
013         gps_ref = np.array([row[1], row[2]])
014
015         # frage ob der Name der Sehenswürdigkeit schon in der Datenbank
016         # existiert. Gib entsprechenden Fehlercode aus wenn wahr.
017         if name == row[0]:
018             return 'err_equal_data'
019
020
021         # frage ob die Sehenswürdigkeit zu nah an einer anderen befindet
022         # Invaliden Radius beträgt dabei 0.00008 Koordinateneinheiten
023         if np.linalg.norm(gps - gps_ref) <= 0.00008:
024             return 'err_equal_data'
025
026     return False

```

Abb. 5: Quellcode der Methode `is_to_near_or_equal_or_not_in_area()`

Die Methode prüft auf Basis der euklidischen Norm des Richtungsvektors beider Koordinaten, ob die zu prüfende Koordinate

1. außerhalb des validen Kartenbereichs liegt (siehe Zeile 7 bis 8),
2. Ob der Name der zu prüfenden Sehenswürdigkeit schon in der Datenbank existiert (siehe Zeile 17 bis 18),
3. ob die Koordinate der Sehenswürdigkeit zu nah an einer der bereits schon in der Datenbank existierenden Sehenswürdigkeiten liegt (siehe Zeile 23 bis 24).

Für den validen Radius, wurde die GPS-Koordinate der Kirche (51.751371, 11.973681) als Mittelpunkt gewählt, sowie eine maximale Koordinatenlänge von 0.005347 (siehe Zeile 7). Für den Mindestabstand einer bereits existierenden Sehenswürdigkeit wurde der Wert 0.00008 Koordinatenlängen festgelegt (siehe Zeile 23).

7. Der Programmablauf zur Erzeugung neuer Sehenswürdigkeiten

Der Programmablauf ist relativ komplex und definiert sich durch den Zusammenschluss verschiedener Skripte auf Basis von PHP und PYTHON, welche HTML-Dokumente erzeugen und Datenbankeneinträge bewirken.

Der Kern basiert auf dem PHP-Skript *php_test.php*, welches die Hauptseite der Webanwendung darstellt und sowohl aus der Karte zum Lokalisieren der Sehenswürdigkeiten als auch einem Upload-Formular für die Erstellung neuer Sehenswürdigkeiten besteht. Sie liegt auf dem Test-Webserver und wird direkt durch die URL:

http://localhost/geodatenbanken_projekt/HTML/php_test.php

abgerufen.

Wenn ein Nutzer die Seite aufruft, wird im oberen Teil etwas Text bzgl. des Seiteninhaltes gezeigt, welcher mit HTML-realisiert wurde. Zudem ist dort ein PHP-Skript integriert, welches das PYTHON-Skript *get_number_of_rows.py* über die Prozessaufruf-Funktion *exec()* aufruft. Das PYTHON-Skript übergibt durch Auslesen der Datenbank die aktuelle Anzahl der Tabellenzeilen zurück. In einem Textabschnitt der Seite wird damit verdeutlicht, wieviel Sehenswürdigkeiten bereits in dem System eingetragen wurden. Das PYTHON-Skript basiert dabei auf dem Methodenabruf der Methode *get_number_of_rows()* aus der Klasse *PY_SQL_API*, welche in der Datei *PY_lib.py* zu finden ist.

Im weiteren Teil der Seite zeigt die Karte den Stadtkern von Köthen, welche im Kapitel 3 bereits erwähnt wurde. Neben den statischen Objekten in der Karte (Legende und ihre Objekte) befinden sich runde Icons an den Stellen, an denen eine Sehenswürdigkeit zu finden ist, welche in der Datenbank aufgenommen wurde (siehe Anhang 1). Dies wird durch einen weiteren PHP-Abschnitt in dem Haupt Quellcode realisiert, welcher über alle Zeilen der Tabelle *Items* in der Datenbank iteriert und die GPS-Koordinaten aus der Zeile ausliest. Die Koordinaten werden an das PYTHON-Skript *gps_to_pxl_transf.py* übergeben, welches die GPS-Koordinaten der jeweiligen Sehenswürdigkeit in Pixelkoordinaten umrechnet und diese an das PHP-Skript zurückgibt. Die Pixel-Koordinaten werden dann in einen HTML-Teil integriert, welcher Hyperlinks in Form von Icons auf die Karte platziert (siehe Kapitel 4). Zudem wird das PYTHON-Skript *make_sub_site.py* aufgerufen und ihm der aktuelle Name der Sehenswürdigkeit übergeben. Dieses Skript erstellt in dem Verzeichnis *sub_html* eine HTML-Datei, welche eine Seite darstellt, auf der das Bild der jeweiligen Sehenswürdigkeit sowie Text zu sehen sind. Diese Seite ist verlinkt mit dem bereits erwähnten Hyperlink. Wenn ein Nutzer auf das Icon klickt, wird er also auf diese Seite verlinkt.

Der Seiteninhalt dieser Sehenswürdigkeiten-Seite wird durch *make_sub_site.py* auf Basis der Methode *get_row_by_name()* aus der Klasse *PY_SQL_API* generiert. *make_sub_site.py* übergibt den Namen der Sehenswürdigkeit an die Methode, welche mit diesem die entsprechende Zeile aus der SQL-Datenbankentabelle ausliest. In dieser stehen sowohl Text, welcher dann auf der Seite zu sehen ist, als auch der Name der Bilddatei, welche durch diese Seite verlinkt ist. Die folgende Abbildung zeigt dies durch den entsprechenden HTML-Quellcode. Dabei wird in einem HTML-Template in Zeile 3 der Name der Sehenswürdigkeit als Titel eingefügt. In Zeile 7 wird der Link zur Bilddatei angepasst und in Zeile 12 wurde der Text eingefügt, welcher dann auf der Seite zu sehen ist.

```
001 <html>
002 <meta charset="UTF-8">
003 <header><titleKirche></></header>
004 <body>
005 <div>
006 <figure>
007 
008 </figure>
009 </div>
010 <div>
011 <p>
012 Die Kirche von Köthen ist wirklich sehr schön.
013 </p>
014 <div>
015 </body>
016 </html>
```

Abb. 6: Exemplarischer HTML-Quellcode der Seite einer Sehenswürdigkeit

In Zeile 7 wird der Variable *src* der Pfad zu dem Bildverzeichnis und dem Namen der Sehenswürdigkeit zugewiesen. In Zeile 11 wird der Informationstext platziert. Aus Platzgründen wurde hier auf übliche Einrückungen vom Quellcode verzichtet.

Im unteren Teil der Hauptseite ist ein Formular implementiert, welches dem Webanwender das Einfügen neuer Sehenswürdigkeiten ermöglicht. Es besteht aus 4 Eingabekomponenten. Die erste Komponente ist ein Datei-Browser, mit dem die Bilddatei zu einer neuen Sehenswürdigkeit zum Hochladen ausgewählt werden kann. Die zweite und dritte Komponente bestehen jeweils aus Texteingabefeldern zum Eingeben der GPS-Koordinaten. Es wird also getrennt zwischen Längen- und Breitengrad. Die letzte Komponente ist ebenfalls ein Texteingabefeld. In diesem kann Text eingefügt werden, welcher die Sehenswürdigkeit beschreibt und dann auf der jeweiligen Seite der Sehenswürdigkeit unter dem hochgeladenen Bild erscheint. Abbildung VI zeigt den HTML-Quellcodeausschnitt des Formulars.

```

001 <form action="/geodatenbanken_projekt/skripte/file_upload.php" me-
thod="post" enctype="multipart/form-data">
002 <p>
003 Bitte wählen Sie genau ein Bild der Sehenswürdigkeit, welches Sie hoch-
laden wollen.
004 </p>
005 <input type="file" name="fileToUpload" id="fileToUpload"><br/><br/>
006 <p>
007 Geben Sie hier die GPS-Koordinaten ein.
008 </p>
009 <label for="latitude">
010 Latitude:
011 <input id="latitude" type="text" name="latitude">
012 </label>
013 <label for="longitude">
014 Longitude:
015 <input id="longitude" type="text" name="longitude">
016 </label>
017 <br/><br/><p>
018 Geben Sie hier eine textliche Beschreibung ein. Am besten schreiben Sie
diese in einem Editor und kopieren diese anschließend hier herein.
019 </p>
020 <label for="info_text">
021 Text:
022 <input id="info_text" type="text" name="info_text">
023 </label>
024 <br/><br/><p>
025 Bitte senden Sie die Daten ab.
026 </p>
027 <input type="Submit" value="Absenden" />
028 </form>

```

Abb. 7: HTML-Quellcodeabschnitt des Eingabeformulars der Hauptseite

Das Formular beginnt in Zeile 1. Dort ist die Verlinkung zu dem PHP-Skript *file_upload.php* definiert, welches die Formulareingaben über die Spezialvariable `$_POST` entgegennimmt. Zeile 5 repräsentiert den Datei-Browser für den Bilddatei-Upload. Zeile 11 und 15 repräsentieren die Eingabemasken zum Definieren der GPS-Koordinate. In Zeile 22 wird die Eingabemaske der Textinfoeingabe definiert. Zeile 27 entspricht einem Button, welcher nach dem Betätigen durch den Anwender die Eingabedaten an den Webserver übermittelt bzw. durch das PHP-Skript entgegengenommen werden kann. Aus Platzgründen wurde hier auf übliche Einrückungen vom Quellcode verzichtet.

Nachdem der Nutzer alle Daten in die jeweiligen Eingabefelder eingegeben und den Absenden-Button gedrückt hat, werden die Daten an den Webserver gesendet und in ein temporäres Verzeichnis gelegt. Zudem wird das PHP-Skript *file_upload.php* aufgerufen, welches über die Spezialvariable `$_POST` Zugriff auf das temporäre Verzeichnis hat. Das Skript hat die Aufgabe, die hochgeladenen Dateien in das Verzeichnis *upload* zu verschieben. Dazu werden der hochgeladenen Infotext sowie die GPS-Koordinate in einer temporären Textdatei namens *data.txt* zusammengefügt (siehe Abbildung VIII).

Die St. Jacobskirche bei Nacht. \$\$token\$\$51.751374\$\$token\$\$11.973689

Abb. 8: Exemplarische Datei bzgl. der Datenübergabe von *file_upload.php* nach *do_files_into_database.py*

Die Daten werden als Textdatei über das Verzeichnis *upload* übergeben. Die Daten werden durch die Text-Token `$$token$$` getrennt. Der erste Teil ist die Texteingabe. Der zweite Teil ist die Latitude. Der letzte Teil ist die Longitude. Das Python-Skript kann somit anhand der Token die einzelnen Daten aus der Textdatei extrahieren.

Anschließend wird das PYTHON-Skript *do_files_into_database.py* aufgerufen. Das Skript liest aus der temporären Textdatei die einzelnen Daten aus und fügt sie über die Methode *insert_into_db()* der Klasse *PY_SQL_API* in die SQL-Datenbank ein, sofern die Daten, geprüft durch die ebenfalls aus dieser Klasse stammenden Methode *is_to_near_or_equal_or_not_in_area()*, als valide gelten.

Valide sind die Daten gemäß der in Kapitel 6 erläuterten Eigenschaften: räumliche Nähe zu bereits eingefügten Sehenswürdigkeiten und Existenz im validen Kartenbereich - also wenn die Sehenswürdigkeit sowohl namentlich als auch räumlich ähnlich noch nicht in der Datenbank existiert.

Sofern die Daten bereits als zu ähnlich gelten, werden sie nicht in die Datenbank übernommen und der Nutzer bekommt durch einen zurückübergebenen Fehler-Code eine Meldung, dass die Koordinate entweder nicht im validen Kartenbereich existiert oder zu nah an einer schon eingetragenen Sehenswürdigkeit ist. Die Rohdaten werden aus dem Verzeichnis *upload* gelöscht.

Andernfalls werden die Daten in die Datenbank eingefügt und ihre Rohdaten aus dem Verzeichnis *upload* gelöscht.

Abbildung IX zeigt das Eingabeformular in gerenderter Form, so wie es der Nutzer sieht.

Sie können gerne eigene Sehenswürdigkeiten hochladen.
Bitte nutzen Sie dazu das folgende Formular.
Beachten Sie, dass nur Sehenswürdigkeiten im Stadtkern von Köthen erlaubt sind.
Es muss die GPS-Koordinate in Dezimalform, sowie ein Bild und eine kurze Beschreibung hochgeladen werden.

Bitte wählen Sie genau ein Bild der Sehenswürdigkeit, welches Sie hochladen wollen.

Keine Datei ausgewählt.

Geben Sie hier die GPS-Koordinaten ein.

Latitude: Longitude:

Geben Sie hier eine textliche Beschreibung ein. Am besten schreiben Sie diese in einem Editor und kopieren diese anschließend hier herein.

Text:

Bitte senden Sie die Daten ab.

Abb. IX: Eingabeformular in gerenderter Form

Die Abbildung zeigt den gerenderten Seitenabschnitt mit dem Dateneingabeformular. Auf ästhetische Formatierung wurde bisher verzichtet. Zu den Eingabefeldern wurde Text implementiert, welcher dem Nutzer Hilfestellung bzw. Regeln stellt, um valide Eingaben tätigen zu können.

Quellen-/Autoren-/Entwicklerverzeichnis

APACHE SOFTWARE FOUNDATION, (1995), *Apache HTTP Server*, Version: 2.4.41, Lizenz: Apache License v2.0, Quelle: Bestandteil von XAMPP

GEOFABRIK, Gründer: TOPF, J., RAMM, F., (2007), *Geofabrik*, Quelle: <http://download.geofabrik.de/europe/germany/sachsen-anhalt.html> , Dateiname: sachsen-anhalt-latest-free.shp.zip, Lizenz: OpenStreetMap-Lizenz

HO, D., (2003), *Notepad++*, Version: 7.8.2, Lizenz: GNU General Public License, Quelle: www.notepad-plus-plus.org

LERDORF, R., GUTMANS, A., SURASKI, Z., (1995), *PHP*, Version: 7.2.27, Lizenz: PHP-Lizenz, Quelle: Bestandteil von XAMPP

MARIADB FOUNDATION, (2009), *MariaDB*, Version: 10.4.2, Quelle: Bestandteil von XAMPP

MICROSOFT, (2015), *Visual Studio Code*, Version: 1.42.1, Lizenz: MIT-Lizenz, Quelle: www.code.visualstudio.com, Quelle: www.qgis.org/de/site/

OLIPHANT, T., (2006), *NumPy*, Zusätzliches Modul zum Lösen linear algebraischer Probleme, Quelle: PIP-Paketmanager von PYTHON

OPENSTREETMAP FOUNDATION, (2004), OpenStreetMap, Quelle: <https://www.openstreetmap.org/relation/2090553#map=17/51.75125/11.97531>

QGIS DEVELOPMENT TEAM, (2002), *QGIS*, Version: 3.4.15-Madeira, Lizenz: GNU General Public License,

SEIDLER, K., VOGELANG, K.; RUEDA, B., LOPEZ RIDRUEJO, D., (2019), *XAMPP*, Version: 7.4.2, Lizenz: GNU General Public License, Quelle: www.apachefriends.org/de/index.html

SUN MICROSYSTEMS (1995 - 2010), MYSQL AB (2010 - 2018), ORACLE CORPORATION (2018), *MySQL*, Version: 8.0.19, Lizenz (aktuell): Teilweise GNU General Public License in Version 2, Teilweise Proprietäre Lizenz durch ORACLE CORPORATION

VAN ROSSUM, G.; (1991), *Python*, Version: 3.7.6, Lizenz: Python-Software-Foundation-Lizenz, Quelle: www.python.org

Anhang

Anhang 1

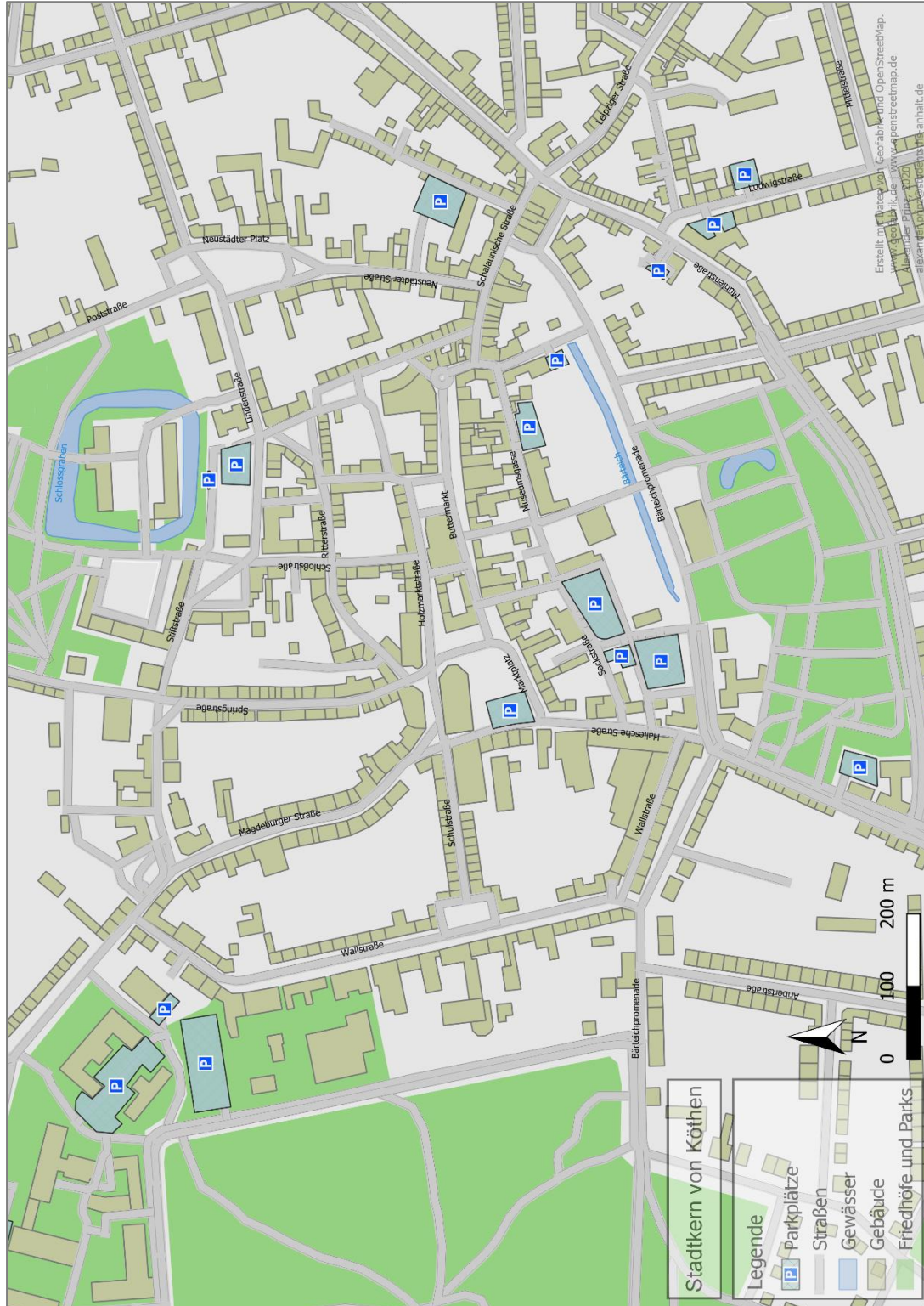
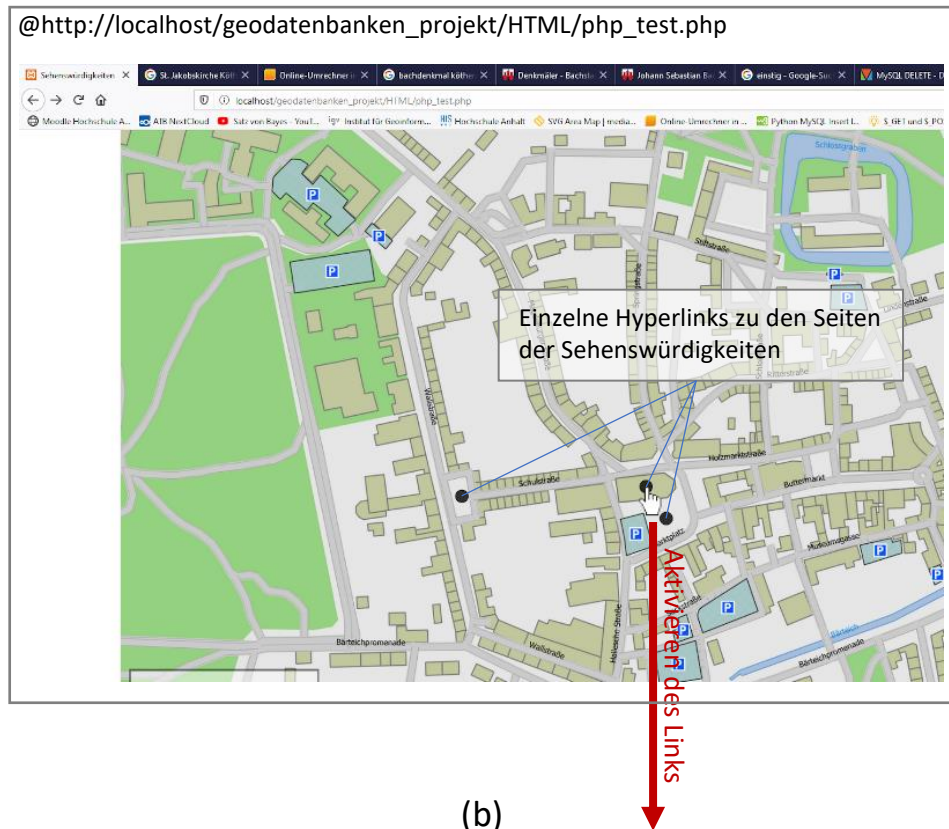


Abb. X: Karte des Stadtkerns von Köthen in vergrößerter Form

Die Abbildung zeigt die gleiche Karte wie in Abbildung 1 in vergrößerter Form.

Anhang 2

(a)



(b)



Abb. XI: Visuelles Schema der Seitenverlinkung

Teil a zeigt einen Ausschnitt aus der Karte mit 3 platzierten Hyperlinks in Form runder schwarzer Icons. Durch Aktivieren eines Hyperlinks mit der Maus öffnet sich die entsprechend dem Icon bzw. Hyperlink zugeordnete Seite der Sehenswürdigkeit (hier am Beispiel der St. Jakobskirche auf dem Marktplatz von Köthen). Auf der Seite ist sowohl das hochgeladene Bild, als auch die Hochgeladene Textinformation zu sehen.