Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 349

Konfigurierbare Softwarekomponente für White-Spot-Analysen

Roman Bitz

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Dipl.-Inf. Falko Kötter, Dipl.-Inf. Sebastian Wagner
Beginn am:	9. Juni 2016
Beendet am:	7. Dezember 2016
CR-Nummer:	D.2.3, E.1, I.3.3

Kurzfassung

Neue Technologien wie Smartphone oder Connected Car tragen stark zum Wachstum von Geodaten bei. Aber auch stationäre Punkte, wie Zulieferer, Produktlager oder Dienstleistungsstationen in Form von Werkstätten oder Ladesäulen wachsen mit neuen Technologien. Es entsteht ein Bedarf an Analysetechniken für Geodaten. Die White-Spot-Analyse hilft dabei, Lücken in Geodaten ausfindig zu machen und stellt eine grundlegende Möglichkeit zur Untersuchung von Geodaten dar.
Diese Arbeit beschäftigt sich mit der Entwicklung einer Softwarekomponente, die Webapplikationen um die Möglichkeit einer White-Spot-Analyse erweitert. Basierend auf OpenStreetMaps und SVG werden drei Visualisierungsformen entwickelt. Geodaten lassen sich auf Abdeckung, Ballung oder nach Postleitzahlverteilung in Deutschland untersuchen. Durch Schnittstellen wie REST oder einer integrierten Weboberfläche wird die Nutzung der Softwarekomponente ermöglicht. Der mögliche Einsatz der Softwarekomponente wird durch eine Evaluation mit realen Werkstattdaten vom Fraunhofer-Institut für Arbeitswirtschaft und Organisation und Ladesäulen vom Rheinisch-Westfälisches Elektrizitätswerk aufgezeigt. Außerdem wird der Nutzen der Softwarekomponente durch einen Produktiveinsatz bestätigt.

Abstract

New technologies like Smartphone or Connected Car are contributing strongly to the growth of geodata. Stationary geopoints, such as suppliers, product stores or service stations in form of car workshops or charging stations are growing with new technologies. There is a need for analytical techniques for those geodata. The white spot analysis helps to find holes in geodata and represents a basic possibility for investigating geodata.
This thesis deals with the development of a software component, which is intended to extend web application with the functionality of a white-spot analysis. Based on OpenStreetMaps and SVG, three forms of visualization are developed. Geodata can be examined for coverage, concentration or postcode distribution in Germany. Through interfaces such as REST or an integrated web interface, the use of the software component is made possible. The possible use of the software component is demonstrated by an evaluation with real car workshop data from Fraunhofer IAO or charging stations from RWE. In addition, the value of the software component is verified by a live deployment.

Inhaltsverzeichnis

1. Einleitung

Mit der Etablierung des Online-Handels wächst der zunehmende logistische Aufwand für die Produktlieferung. Um Lieferzeiten einhalten zu können, werden möglichst kurze Verbindungen zwischen verschiedenen Standorten benötigt. Eine gute Abdeckung von z.B. Lagerhallen ist hier essentiell. Solche geographischen Verteilungen spielen auch in anderen Bereichen eine wichtige Rolle. Die bereits etablierte Telekommunikationsbranche muss sich um die Verteilung von Funkmasten und die Nachrüstbarkeit auf neue Standards wie LTE kümmern [Neu15]. Aber auch das Wachstum der Elektromobilität verlangt eine gute Abdeckung der Ladeinfrastruktur in Deutschland [NPE15].
Ein Teil der Forschungsarbeit beim Fraunhofer-Institut für Arbeitswirtschaft und Organisation beschäftigt sich mit der Kfz-Branche. Eine häufig auftretende Herausforderung ist dabei die räumliche Verteilung von Angeboten zu untersuchen und potenzielle Lücken zu entdecken. Diese Lücken werden auch White Spots genannt. Ein konkretes Beispiel ist die Verteilung von Kfz-Werkstätten in Deutschland, welche nach Prüfung von bestimmen Qualitätsmerkmalen, Geschädigten von Versicherungen empfohlen werden können. Eine solche White-Spot-Analyse führt zu einem nicht unerheblichen Aufwand. Eine anschließende Aufbereitung und Visualisierung der Ergebnisse muss bisher manuell getätigt werden.

1.1. Aufgabenstellung

Ziel dieser Arbeit ist es, eine konfigurierbare Softwarekomponente für White-Spot-Analysen zu entwickeln. Die Softwarekomponente soll dem Nutzer eine visuelle Darstellung über die gegebenen Geodaten bieten, als auch mögliche Lücken in den Daten anzeigen. Die Geodaten sollen über ein Standardformat einlesbar sein. Die Bearbeitung der Daten soll durch Parameter wie z.B. Abdeckungsradius konfigurierbar bleiben. Die Ergebnisse der Softwarekomponente sollten vorwiegend visuell vorliegen. Bei der Visualisierung sollen Landkarten zum Einsatz kommen. Durch verschiedene Überlagerungsmöglichkeiten wie z.B. Heatmaps soll die Abdeckung dargestellt werden. Nach Fertigstellung der Softwarekomponente soll die Funktionalität über ein Szenario mit realen Daten des Fraunhofer IAO verifiziert werden.

Für die Durchführung der Arbeit sind folgende Teilschritte vorgesehen. Zunächst sollen die Anforderungen an die Softwarekomponente festgehalten werden. Die Anforderungen bieten eine Grundlage zur Vorstellung der fertigen Softwarekomponente. Aus dieser Grundlage soll eine Recherche über nötige theoretische Grundlagen bezüglich der White-Spot-Analyse getätigt

werden. Anschließend sollten Standardformate, Schnittstellen und Konfigurationsparameter festgelegt werden, welche dann als Grundlage für die Konzeption der Architektur und Nutzerschnittstellen der Softwarekomponente bereitstehen. Die anschließende Implementierung der Softwarekomponente soll Datenaufnahme, Analysealgorithmen, Ergebnisvisualisierung und die Nutzerschnittstellen enthalten. Eine Form der Dokumentation ist Teil der Implementierung. Abschließend soll eine White-Spot-Analyse anhand eines realen Szenarios in der Kfz-Branche mit der Softwarekomponente durchgeführt werden.

1.2. Gliederung

Die Arbeit ist in 7 Kapitel aufgeteilt.
Kapitel 1 gibt eine Einführung in das Thema und stellt die Aufgabestellung dar. Kapitel 2 formuliert aus der Aufgabestellung sieben Anforderungen an die Softwarekomponente. Kapitel 3 ist eine Einführung in die Terminologie. Außerdem werden genutzte Verfahren mit Geodaten erklärt. Kapitel 4 beschäftigt sich mit Geodaten und ihren Eigenschaften. Dabei wird auf Qualitätsmerkmale, sowie die Dimensionen von Geodaten eingegangen. In diesem Abschnitt werden auch zwei standardisierte Dateiformate angeschaut. Kapitel 5 stellt den Softwareentwurf von WASC vor und erläutert Funktionalitäten der Softwarekomponente. Weiterhin werden auch alle Ausgabemöglichkeiten und Konfigurationsparameter erläutert. Teil dieses Kapitels sind auch Nutzerschnittstellen, sowie die genutzten Bibliotheken. Kapitel 6 führt mit der fertigen Softwarekomponente eine Evaluation mit drei Anwendungsszenarien durch. Dazu werden Datensätze von der Softwarekomponente analysiert und anschließend evaluiert. Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und stellt einen Ausblick über weitere Anwendungen von Geodaten.

2. Anforderungen

Vor Beginn jeder Softwareentwicklung ist die Klarstellung der Anforderungen eine wichtige Grundlage. Das Ziel dieser Bachelorarbeit ist die Entwicklung einer konfigurierbaren Softwarekomponente für White-Spot-Analysen. Dazu wird vorab geklärt welche Anforderungen an die Software gestellt werden. Grundlage für die Anforderungskriterien bietet die Aufgabestellung aus Kapitel 1.1.

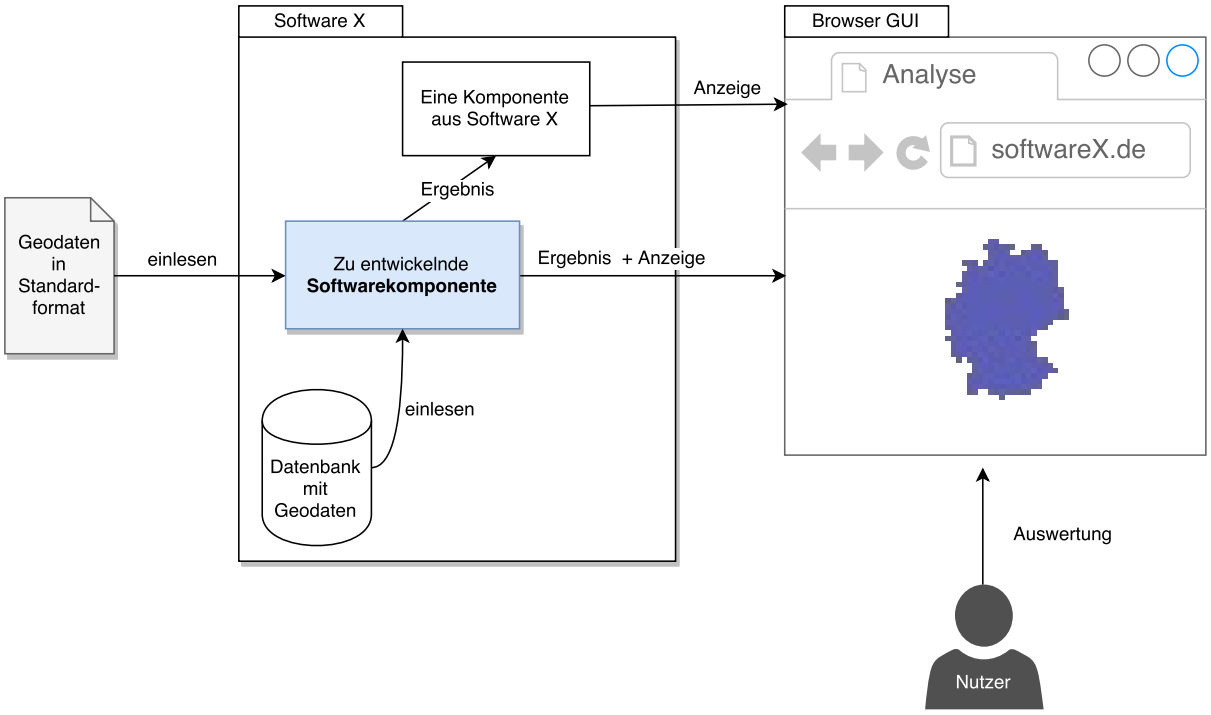

Abbildung 2.1.: Einsatzzweck der Softwarekomponente. Eigene Abbildung.

In der Abbildung 2.1 sieht man einen möglichen Einsatzzweck für die Softwarekomponente. Die Softwarekomponente ist als hellblaue Entität in der Abbildung zu sehen. Die Komponente kann Geodaten sowohl in einem Standardformat, als auch aus einer Datenbank einlesen. Die Komponente ist in eine andere 'Software X' integriert und liefert Ergebnisse an eine Komponente von 'Software X'. 'Software X' stellt hierbei eine beliebige Webapplikation dar. Außerdem kann die Komponente Ergebnisse direkt im Browser des Nutzers anzeigen. Der Nutzer wertet die Ergebnisse aus.

2.1. Anwendungsszenario

Die Softwarekomponente soll anwendungsfallübergreifend und wiederverwendbar funktionieren. Der erste reale Einsatz soll die Kfz-Branche in Deutschland sein. Ein mögliches Beispiel ist die geografische Verteilung von Autowerkstätten. Mit der bereits anbahnenden Elektromobilität wäre die geografische Verteilung von Ladesäulen auch ein mögliches Szenario. Nicht direkt mit der Kfz-Branche zusammenhängend, aber dennoch ein ähnliches Anwendsszenario bietet die Telekommunikation. Die Abdeckung von Funktürmen und Verfügbarkeit von 2G, 3G oder LTE-Netzen [Vod16] wäre ein möglicher Anwendungszweck für die Softwarekomponente.
Diese Anwendungsszenarien basieren auf geeigneten Geodaten (Kapitel 4). Dementsprechend muss die Softwarekomponente Geodaten einlesen können. Die Beschaffung solcher Daten ist jedoch nicht Teil der Softwarekomponente.

2.2. Geodaten einlesen

Für die Verarbeitung der Geodaten muss die Softwarekomponente diese einlesen können. Das Einlesen sollte in einem Standardformat möglich sein. Die Formatierung von ungeordneten Datensätzen ist nicht Teil der Softwarekomponente. Es sollte jedoch ein Datenmodell vorliegen mit welchem eine korrekte Formatierung möglich ist. Für die Kompatibilität mit anderen Softwareprojekte sollte eine Datenimport Funktionalität bestehen. Diese kann z.B. durch eine Datenabfrage vorliegen. Der Import kann sowohl manuell getätigt werden, als auch automatisierbar sein.

2.3. Schnittstellen

Bei einer Softwarekomponente handelt es sich um ein Erweiterungsmodul für andere Softwaresysteme. Daraus lässt sich schließen, dass die Softwarekomponente nicht allein lauffähig sein muss, sondern andere lauffähige Softwareprojekte benötigt. Ein solches Erweiterungsmodul muss eine Schnittstelle bereitstellen. Dabei sollte es sowohl eine Programmschnittstelle geben, welche durch reine Funktionsaufrufe funktioniert, als auch eine Nutzerschnittstelle, welche der Nutzer durch eine GUI (Graphical User Interface) benutzen kann.

2.4. Visualisierung der Ergebnisse

Die Ausgaben der Softwarekomponente werden von Menschen ausgewertet, deswegen müssen alle Ergebnisse visuell vorliegen. Dafür sollten verschiedene Visualisierungsmöglichkeiten bereitstehen. Die Ausgaben können teilweise Interaktiv sein um den Nutzer bei der Auswertung zu Unterstützen. Hierfür können geografische Karten wie OpenStreetMaps oder entsprechende

SVGs (Scalable Vector Graphics) [Moz15] genutzt werden. Die Ergebnisse müssen in eine Weboberfläche integrierbar sein.

2.5. Analyse-Algorithmen

Die Hauptaufgabe der Softwarekomponente soll Lücken in Geodaten aufdecken. Für ein solches Vorgehen werden verschiedene Algorithmen benötigt. Die Softwarekomponente sollte alle nötigen Algorithmen bereitstellen um eine erfolgreiche White-Spot-Analyse durchzuführen. Sollten bereits etablierte Algorithmen nicht ausreichend sein, so sollen eigene Vorgehensweisen entwickelt werden.

2.6. Ergebnisauswertung

Die Auswertung der Ergebnisse ist nicht Teil der Softwarekomponente. Die entsprechende Evaluation soll vom Nutzer selbst getätigt werden. Die Softwarekomponente muss keine Bewertungen oder Vorschläge anhand der Ergebnisse abgeben. Jedoch kann eine Unterstützung durch Zoom-Funktionen, Tooltips oder farbliche Konfiguration der Ergebnisse vorliegen.

2.7. Evaluation der Software

Nach Fertigstellung der Softwarekomponente soll eine Evaluation über die Funktionalität der Softwarekomponente erfolgen. Dazu wird die Komponente im Zusammenhang mit dem Datenbestand am Fraunhofer IAO evaluiert. Der Datenbestand besteht aus in Deutschland befindlichen Autowerkstätten und deren geografische Koordinaten.

3. Grundlagen

Eine White-Spot-Analyse beinhaltet Fragestellungen aus den Gebieten der Geografie, geografischen Systemen und dazugehörigen Algorithmen. Diese Themenbereiche bringen nicht nur eigene Begriffe, sondern auch bereits bekannte Vorgehensweisen mit sich. Dieses Kapitel beschäftigt sich mit den Grundlagen, die im späteren Verlauf für die Entwicklung der Softwarekomponente nützlich sein können. Englische Begriffe werden in ihrer Sprache belassen, da der Umgang mit ihnen in der Softwaretechnik üblich ist.

3.1. White-Spot-Analyse

Die Begrifflichkeit von White-Spots bzw. 'weißen Flecken' wurde schon vor Jahrhunderten in der Kartographie verwendet. Damals waren 'weiße Flecken' unerforschte Gebiete, zu welchen es keine kartographischen Aufzeichnungen gab [Fra09]. Heutzutage spricht man von Bedarfserkennung. Das heißt es geht um die Erkennung von geografischen Positionen, welche von einem bestimmten Angebot nicht bedient werden [Wü16]. Ein Beispiel dafür wäre der Bau von neuen Unternehmensstandorten. Vor dem Bau eines neuen Standortes, werden entsprechende geografische Lagen nach verschiedenen Erfolgsfaktoren analysiert. In so einem Fall will ein Unternehmen nicht in einem White-Spot landen, welcher z.B. eine schlechte Hochschulstruktur oder schlechte logistische Anbindungen aufweist [Ber15]. Mit der White-Spot-Analyse lassen sich also Lücken in einem geografischen Raum ausfindig machen. Aus der technischen Sicht betrachtet werden für so ein Verfahren Algorithmen verwendet, um die Verteilung von Geodaten festzustellen und daraus Lücken in bestimmten Gebiet aufzuzeigen.

3.2. GIS-Software

GIS steht für **G**eo**I**nformations**S**ystem. GIS-Software ist ein Programm das mit Geodaten im geografischen Raum arbeitet. Die Hauptaufgabe einer GIS-Software ist es dem Nutzer den Umgang mit Geodaten zu vereinfachen. Dazu gehört auch der visuelle Aspekt. Geodaten können grafisch angezeigt werden und z.B. Straßen, Flüsse oder Land darzustellen. Ein Modellierungstool wie z.B. CAD ist dabei keine GIS-Software [Pro05]. Die Softwarekomponente dieser Arbeit ist auch als GIS-Software zu sehen.

3.3. Geospatial

Geospatial ist ein englischer Begriff, welcher oft genutzt wird um bereits etablierte Begriffe dem geografischen Kontext zuzuweisen [Dr.08]. *Geospatial Analysis* würde georäumliche Analyse bedeuten und *Geospatial Data* entsprechend Geodatensätze heißen. Eine White-Spot-Analyse wäre eine *Geospatial Analysis* und GeoJSON (Kapitel 4.4.2) wäre ein Standardformat für *Geospatial Data*.

3.4. Orthodrome

Orthodrome ist die kürzeste Verbindung zwischen zwei Punkten auf der Erdoberfläche [Sch08]. Die Ermittlung der Distanz zwischen zwei Punkten ist ein Grundwerkzeug in jedem geografischen Softwaresystem. Da sich die Distanzberechnung rein mathematisch durchführen lässt, ist der Einsatz nicht nur in Programmiersprachen möglich, sondern auch in Abfragesprachen wie SQL durchführbar [Ava16].
Das Besondere an einer Distanzberechnung zwischen zwei geografischen Koordinaten, ist die Erdkrümmung. Zeichnet man eine Linie auf die Erdkugel, so ist dieser nicht zweidimensional flach in einer Ebene, sondern eine gewölbte Linie. Deswegen muss bei der Distanzberechnung auch die Form der Erdkugel in Betracht gezogen werden. Gerechnet wird in Kilometern. Zunächst benötigt man den Erdradius. Dieser beträgt ca. 6371km [Pro01]. Anschließend werden zwei Punkte *P* und *Q* benötigt, von welchen der Abstand berechnet werden soll. Beispielhaft sei der Breitengrad *Plat* = 48.78 und der Längengrad *Plng* = 9.17 vom Punkt *P* und der Breitengrad *Qlat* = 50.43 und der Längengrad *Qlng* = 80.18 vom Punkt *Q* gegeben.
Normalerweise werden in Programmiersprachen die Winkel als Bogenmaß berechnet. Also sollte vorher der Längen- und Breitengrad in den jeweiligen Radianten umgewandelt werden. Der Radianten aus einem Grad wird wie folgt berechnet [Ber64].

$$Grad * (\pi/180) = Radiant$$

Für die beiden Punkte *P* und *Q* sieht das wie folgt aus.

$$P : 48.78 * (\pi/180) = 0.851372$$
$$9.17 * (\pi/180) = 0.160047$$

$$Q : 50.43 * (\pi/180) = 0.880170$$
$$80.18 * (\pi/180) = 1.39940$$

Somit beträgt der Radiant vom Breitengrad *Plat* = 0.851372, vom Längengrad *Plng* = 0.160047, der Radiant vom Breitengrad *Qlat* = 0.880170 und vom Längengrad *Qlng* = 1.39940. Anschließend muss man die Werte in die folgende Gleichung einsetzen [Mar16].

Formel:

$$Erdradius * acos(sin(Plat) * sin(Qlat) + cos(Plat) * cos(Qlat) * cos(Qlng - Plng))$$

Eingesetzt:

$$6371 * acos(sin(0.851372) * sin(0.880170) + cos(0.851372) * cos(0.880170) * cos(1.39940 - 0.160047))$$

Das Ergebnis beträgt 4919.34 km.

3.5. Umkreissuche

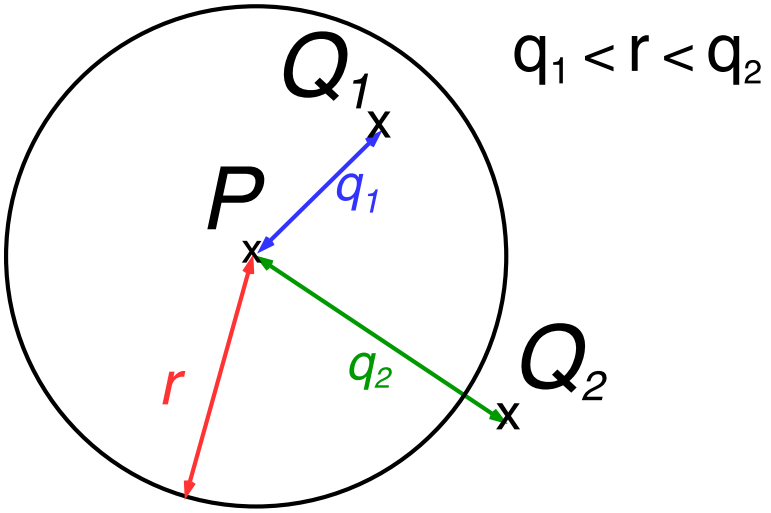

Abbildung 3.1.: Darstellung vom Umkreis um Punkt *P* mit zwei Punkte *Q1* und *Q2*. Die Längen der farbigen Geraden (*r*, *q1*, *q2*) sind wie folgt geordnet '*q1* kleiner *r* kleiner *q2*'. Eigene Abbildung.

Mit Hilfe der Orthodrome kann man auch eine Umkreissuche durchführen. Dazu wird ein Radius *r* gewählt, welcher den Umkreis um Punkt *P* bestimmt. Liegt ein Punkt *Q* innerhalb des Umkreises von Punkt *P*, so ist der Abstand zwischen *P* und *Q* kleiner gleich *r*. Das heißt für die Feststellung ob ein Punkt im Umkreis eines anderen Punktes liegt, reicht es den Abstand zwischen diesen zu berechnen und mit dem gegebenen Radius abzugleichen. Die Abbildung 3.1 veranschaulicht den Gedankengang. Die Umkreissuche kann dazu genutzt werden um z.B. festzustellen ob Geodaten sich in einem speziellen Gebiet befinden.

3.6. Geocoding

Mit Geocoding wird ein Verfahren bezeichnet in dem eine Adresse, Ortsbeschreibung oder Postleitzahl einem geografischen Raum zugewiesen wird [Sch10]. Dieses Verfahren ist oft bei Kartendiensten wie Google Maps[1] oder OpenStreetMaps[2] zu beobachten. Durch Eingabe von z.B. 'Uni Stuttgart', werden geografische Koordinaten ausgegeben.
Die Datenstruktur von elektronischen Karten basiert auf der Graphentheorie [Gol07]. Landkarten werden mit Knoten und Kanten modelliert. Den Knoten und Kanten werden verschiedene Attribute zugeordnet. Die Attribute helfen bei der Abbildung der Datenstruktur auf die reale Welt. Die geografischen Koordinaten von Knoten oder Kanten können anschließend für das Geocoding genutzt werden.
Ein Geocoding Algorithmus lässt sich in zwei Teile aufteilen. Es gibt das *Feature Matching* und das *Feature Interpolation*. *Feature Matching* soll die Verbindung zwischen der gesuchten Eingabe und dem dazugehörigen Bereich finden. Während *Feature Interpolation* versucht den exakten Knoten zu der Eingabe zu finden [Gol07].
Das Kernelement beim *Feature Matching* ist die Umsetzung einer Referenzierung von Eingaben zu Knoten. Für die Referenzierung kann z.B. eine Datenbank verwendet werden. In der Datenbank lassen verschiedene Begriffe als Referenz an Knoten knüpfen. Zunächst wird eine Eingabe geprüft und analysiert. Fünfstellige Nummer können dabei z.B. auf eine Postleitzahl hindeuten, während Abkürzungen wie 'str' sehr wahrscheinlich auf einen Straßennamen zeigen. Anschließend wird eine Datenbankanfrage durchgeführt. Die enthaltenen Knotenreferenzen sollten auf einen Bereich im Graph zeigen. Die Qualität einer Geocodierung ist damit sehr stark von der Qualität der Datenbank abhängig.
Mit *Feature Interpolation* wird ein bereits referenzierter Bereich im Graph interpoliert um eine exakte Stelle ausfindig zu machen. Dafür werden die beiliegenden Informationen in den Knoten und Kanten genutzt. Enthielt die Eingabe z.B. eine Straßennummer, die aber nicht im Graphen hinterlegt ist, so kann über den die Länge der Straße eine ungefähre Position ausgegeben werden. Vorausgesetzt es sind andere Straßennummern im Graphen hinterlegt [Gol07].

[1] https://www.google.com/maps
[2] https://www.openstreetmap.org/

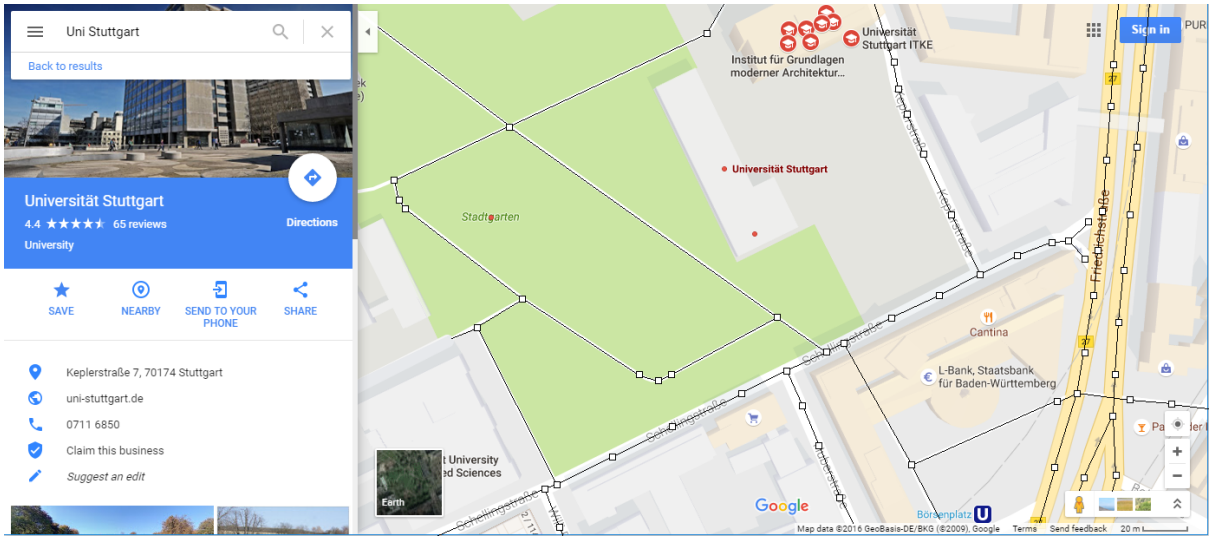

Abbildung 3.2.: Ergebnis einer Google Maps Suche (https://www.google.de/maps/) nach der Eingabe 'Uni Stuttgart'. Die Abbildung zeigt auch einen Graphen für das Straßensystem. Dieser Graph wurde zur Veranschaulichung nachträglich eingefügt. Er spiegelt nicht den originalen Graphen von Google Maps wieder.

4. Geodaten

Eine White-Spot-Analyse arbeitet mit Geodaten, die einem geografischen Raum zugeordnet sind. Aus diesem Grund befasst sich dieses Kapitel mit Geodaten und ihren Eigenschaften. Dabei werden die Qualitätsmerkmale von Geodaten angeschaut, die verschiedenen Dimensionen in den diese vorliegen können, offene Dateiformate als auch die Definition.

4.1. Definition

Das Europäische Parlament und der Rat der Europäischen Union hat am 25.4.2007 Richtlinien zur Schaffung einer Geodateninfrastruktur in der Europäischen Gemeinschaft (INSPIRE) veröffentlicht. Geodaten werden dort in Artikel 3 Nr. 2 wie folgt klassifiziert. [DAS07]

> „Geodaten“ alle Daten mit direktem oder indirektem Bezug zu einem bestimmten Standort oder geografischen Gebiet

Daraus kann man schließen, dass Geodaten nicht nur als eine direkte Beschreibung von Objektpositionen gesehen werden, sondern auch einen indirekten Bezug zum Objekt darstellen können. Ein Beispiel für einen indirekten Bezug wäre die Herleitung eines geografischen Bereichs, durch eine Postleitzahl. Weiterhin sind Geodaten an ihren Raum gebunden. So sind geografische Geodaten nicht unbedingt mit Geodaten aus einem Simulationsprogramm vergleichbar.

4.2. Dimensionen

Geodaten können in vier verschiedenen Dimensionen vorliegen [Pro10a]. Dabei werden die unterschiedlichen Dimensionen für verschiedene geografische Zwecke genutzt.

4.2.1. Zweidimensionale Geodaten (2D)

Zweidimensionale Geodaten besitzen zwei Koordinaten pro Punkt, eine X- und Y-Koordinate. Dadurch lassen sich die Punkte in ein zweidimensionales Koordinatensystem zeichnen [Ins03]. Zweidimensionale Geodaten finden vor allem in der Geographie Anwendung. Stadt- oder

Landkarten sind in der Vogelperspektive gedruckt und liefern ein Raster mit alphanumerischen Koordinaten (Abbildung 4.1). Ein standardisiertes Verfahren von zweidimensionale Geodaten, sind geographische Koordinaten. Diese werden in Breitengrad und Längengrad angegeben. Dabei wird die dreidimensionale Welt auf ein zweidimensionales Koordinatensystem abgebildet. Das geschieht durch wegstreichen von Höhen- und Tiefen-Parameter. Dafür wurde die Erdkugel in 180 Breitengrade und 360 Längengrade aufgeteilt. Ein Punkt besteht also aus der Angabe eines Breiten- und Längengrades. In der Informatik wird die Dezimalschreibweise der Grade benutzt. 48.745710, 9.106174 wäre ungefähr der Standort von der Universität Stuttgart in Vaihingen [1]. Durch die Anzahl der Nachkommastellen lässt sich die Genauigkeit der Daten regulieren.

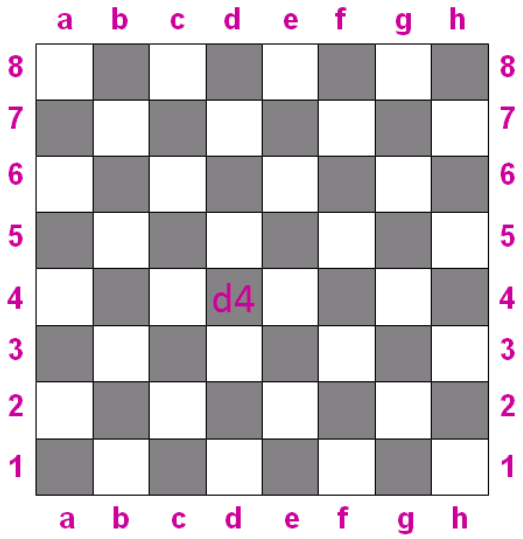

Abbildung 4.1.: Ein Schachbrett mit alphanumerischen Koordinaten. Jedes Kästchen auf dem Schachbrett lässt sich mit einer Kombination aus Buchstabe und Zahl eindeutig identifizieren [Gie06].

4.2.2. Zweieinhalbdimensionale Geodaten (2,5D)

Zweieinhalbdimensionale Geodaten haben wie schon zweidimensionale Geodaten eine X- und Y-Koordinate pro Punkt zugeordnet, jedoch wird zu jedem (X, Y) Paar ein weiterer Wert H hinzugefügt. Dabei kann H jegliche Art von Information enthalten.
Betrachtet man H als ein Höhenwert, welcher jeder geographische Punkt zugeordnet ist, so kann man aus der neugewonnen Information eine Topografische Karte generieren [Ins03].
Zweieinhalbdimensionale Geodaten sind ein wichtiger Bestandteil in der Informatik, wenn es um Arbeit mit zweidimensionalen Geodaten geht. Die Zuordnung von Zusatzinformationen

[1] https://www.google.com/maps/place/48.7455947,9.0874446

pro Punkt erlaubt erst die Verarbeitung und Analyse der Daten. Man kann z.B. jedem Punkt eine Gewichtung zuordnen. Daraus ergeben sich neue Möglichkeiten in der Visualisierung. Zum Beispiel kann man Punkte nicht nur anzeigen, sondern je nach Gewichtung einfärben oder die Größe ändern.

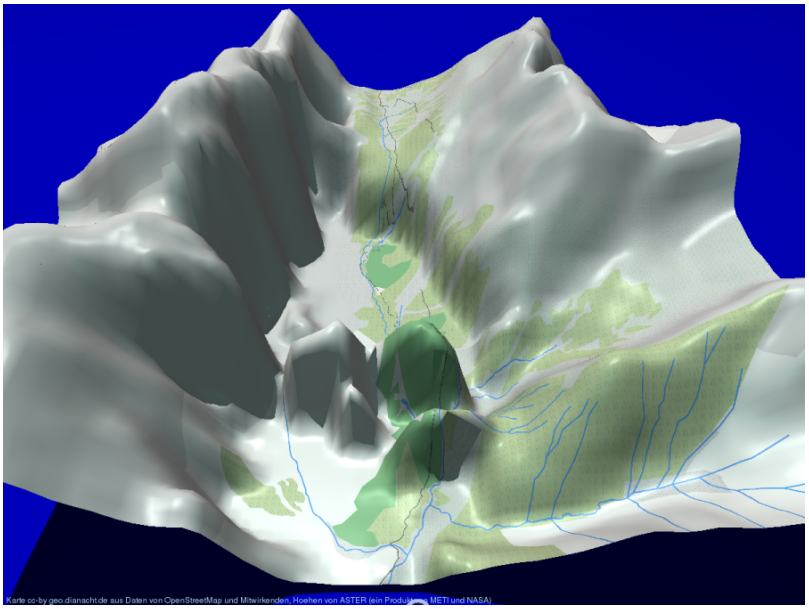

Abbildung 4.2.: Darstellung eines Digitalen Höhenmodells mit 2,5-dimensionalen Geodaten. Das Modell wird in einem 3-dimensionalen Raum visualisiert. Dabei sieht man gut, dass es sich um eine hohle Fläche handelt, da die Beschränkung des H Parameters keine Darstellung eines Körpers erlaubt [Ber13].

4.2.3. Dreidimensionale Geodaten (3D)

Dreidimensionale Geodaten liegen als dreiwertiges Paar vor. Man hat eine X-, Y- und Z Koordinate pro Punkt zugeordnet. Alle drei Werte sind an den Wertebereich ihres Raumes gebunden. Eine Einschränkung wie bei zweieinhalbdimensionalen Geodaten (ein H Wert pro (X, Y) Paar) gibt es nicht [Ins03]. Dreidimensionale Geodaten sind vor allem in der Modellierung von computergenerierten 3D Objekten zu finden. In der Geografie findet eine solche Modellierung für Gebäudekörper statt, um dreidimensionale Karten zu erstellen. Auf gleicher Thematik basieren auch CAD Programme, mit denen Ingenieure Maschinenteile für die Herstellung designen.

Abbildung 4.3.: Modellierung von New York durch dreidimensionale Geodaten. Diese Abbildung ist eine Momentaufnahme aus Google Earth [Goo16a].

4.2.4. Vierdimensionale Geodaten (4D)

Vierdimensionale Geodaten liegen als vierwertiges Paar vor. Dabei wird die vierte Koordinate T, als zeitliche Dimension gesehen. Man hat somit zu jedem Punkt (X, Y, Z) auch einen Zeitstempel T gegeben. Damit lassen sich alle dreidimensionalen Objekte auch in ihrem zeitlichen Verlauf betrachten. Zu jedem Zeitschritt lässt sich die Existenz oder Nichtexistenz eines (X, Y, Z) Paars zuordnen [Tua12].
Vierdimensionale Geodaten werden auch in der Luftfahrt verwendet. Während die Position des Flugzeugs in der Luft dreidimensional bestimmt wird, kann man durch die zeitliche Dimension den Flugverlauf erkennen. In der Abbildung 4.4 sieht man, wie vierdimensionale Geodaten von Flugzeuglinien realisiert werden können. Das Projekt Flightradar24 [AB16] zeigt in Echtzeit Flugbahnen von Flugzeugen. Flugbahnen lassen sich auch in ihrem zeitlichen Verlauf betrachten.

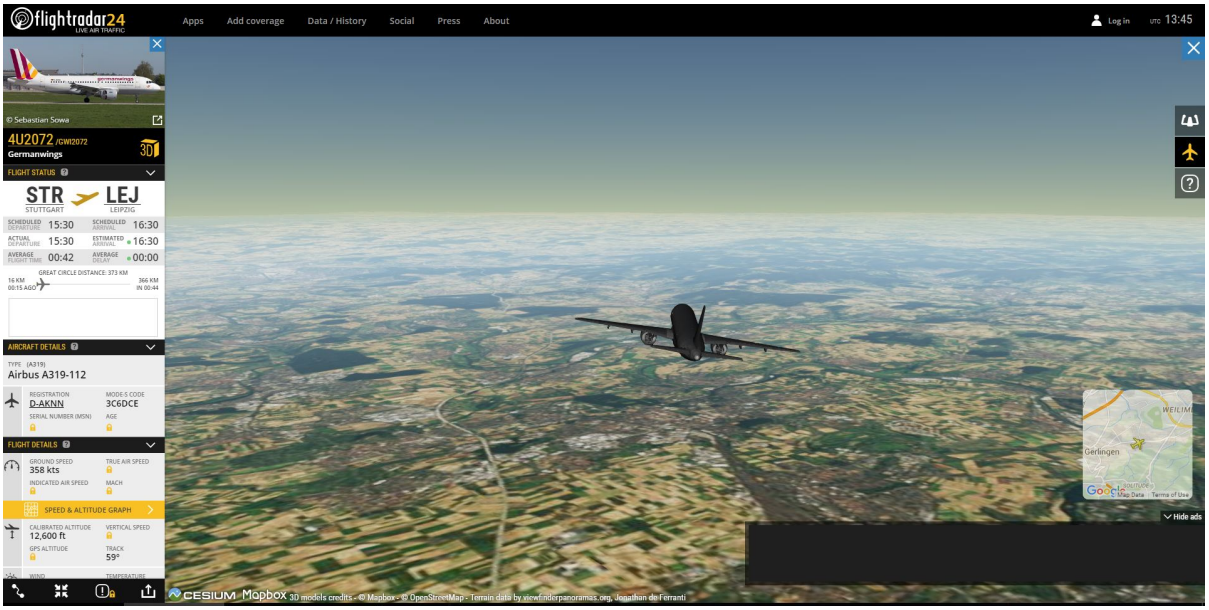

Abbildung 4.4.: Verbindung zwischen 2-dimensionaler Karte und 4-dimensionalen Flugzeugdaten. Diese Abbildung ist eine Momentaufnahme aus Flightradar24 [AB16].

4.3. Qualität

Die Qualität von Geodaten hat eine Auswirkung auf das Ergebnis einer White-Spot-Analyse. Mit qualitativ schlechten Daten lassen sich schlechte oder gar falsche Ergebnisse erzielen. Deswegen ist eine Aufstellung von Qualitätsparametern nötig. In der ISO 19157:2013 Abschnitt Annex D werden 5 Qualitätsparameter für Geodaten genannt. [Int13]

4.3.1. Vollständigkeit

Unter Vollständigkeit versteht man die reine Belegung von Werten. Hierbei muss festgestellt werden ob alle Attribute des Datensatzes mit Werten belegt sind oder diese leer (NULL) sind. Weiterhin ist darauf zu achten, ob es einen Datenüberschuss oder Datenmangel gibt. Bei einem Datenüberschuss liegen meist zusätzliche Attribute im Datensatz vor oder es existieren Zusatzinformationen. Bei einem Datenmangel fehlen Attribute oder Datensatzteile [Pro10b] [Joo00] [Pro05].

4.3.2. Logische Konsistenz

Bei der logischen Konsistenz geht es vor allem um korrekten Syntax. Es muss eine korrekte Formatierung der Daten vorliegen. Der Wertebereich aller Attribute darf nicht überschrit-

ten werden und alle räumlichen Regeln müssen erfüllt sein. Mit räumlichen Regeln ist die Konsistenz der Geodaten in ihrem jeweiligen Raum gemeint. Bei einem Modellierten Objekt wären das die geometrischen Regeln. In der Geographie wären das Nachbarschaftsbeziehungen [Pro10b] [Joo00] [Pro05].

4.3.3. Positionsgenauigkeit

Die Positionsgenauigkeit beschreibt den Unterschied zwischen Geodatenwert und Realwert. Dabei muss überprüft werden, wie stark ein Datensatz vom Realwert abweicht. Es müssen sowohl absolute Unterschiede, als auch relative Unterschiede überprüft werden. Dazu muss auch festgestellt werden wie weit sich das digitale Raster vom Realraum unterscheidet [Pro10b] [Joo00] [Pro05].

4.3.4. Zeitliche Genauigkeit

Zeitliche Genauigkeit ist für vierdimensionale Geodaten (Kapitel 4.2.4) sehr wichtig. Dafür muss die Auflösung (Minutengenau, Sekundengenau, ...) des Zeitstempels validiert werden. Auch die Richtigkeit der zeitlichen Konsistenz, also die Reihenfolge der zeitlichen Ereignisse muss korrekt sein. Außerdem ist die Gültigkeit des Zeitstempels zu validieren. Damit ist gemein, inwieweit ist ein Zeitstempel X gleich einem Zeitstempel Y ist, wenn zu Y kein exakter Datensatz vorliegt [Pro10b] [Joo00] [Pro05].

4.3.5. Thematische Genauigkeit

Bei der Thematische Genauigkeit geht es um die Feststellung, ob die digitalen Objekte von Geodaten auf die richtigen Klassen des Realraums abbilden. Das heißt die digitale Abbildung einer Straße soll auch auf eine Straße in der realen Welt abbilden. Eine falsche Abbildung wäre, wenn die digitale Straße auf einen realen Fluss abbilden würde. Die Thematische Genauigkeit ist besonders für Navigationssysteme und Militär wichtig, da in diesen Bereichen eine falsche Abbildung schwere Folgen haben könnte [Pro10b] [Joo00] [Pro05].

4.4. Dateiformate

In der Informatik werden Information bzw. Daten häufig in standardisierten Dateiformaten ausgetauscht. Zwei bekannte Dateiformate sind XML (E**x**tensible **M**arkup **L**anguage) [Tim06] und JSON (**J**ava**S**cript **O**bject **N**otation) [jso16]. Speziell für Geodaten gibt es sehr viele Dateiformate. Das liegt insbesondere auch an verschiedenen GIS-Systemen. Basierend auf XML und JSON werden in diesem Abschnitt die beiden Geodatenformate KML und GeoJSON vorgestellt.

4.4.1. KML

KML steht für **K**eyhole **M**arkup **L**anguage und ist ein primär für Google Earth entwickeltes XML Format [Goo16b]. In KML kann man geografische Koordinaten, Zeiträume, Kartendesign sowie Blickwinkel angeben. Durch diese Flexibilität kann KML sowohl für Geografische Zwecke als auch teilweise für Modellierung genutzt werden.
Für eine White-Spot-Analyse wäre hier der Tag *<Placemark>* interessant. *<Placemark>* kann geometrische Objekte enthalten wie z.B *<Point>*. Weiterhin kann auch der Tag *<ExtendedData>* enthalten sein. In *<ExtendedData>* lassen sich *<Data>* Tags einfügen die beliebige Werte enthalten können. Mit *<Placemark>* kann man also Punkte mit beliebigen Zusatzinformationen erzeugen [Goo16c]. In einer White-Spot-Analyse kann die Möglichkeit der Zusatzinformation für wichtige Parameter und Eigenschaften des jeweiligen Punktes genutzt werden. Dadurch können Analysen detaillierter werden, da qualitativ bessere Daten vorliegen.

Ein Beispiel KML Dokument mit einem Punkt und Linie:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <kml xmlns="http://www.opengis.net/kml/2.2">
3    <Document>
4      <Placemark>
5        <ExtendedData>
6          <Data name="prop">
7            <value>value</value>
8          </Data>
9        </ExtendedData>
10       <Point>
11         <coordinates>102,0.5</coordinates>
12       </Point>
13     </Placemark>
14     <Placemark>
15       <ExtendedData>
16         <Data name="propone">
17           <value>value</value>
18         </Data>
19         <Data name="proptwo">
20           <value>0</value>
21         </Data>
22       </ExtendedData>
23       <LineString>
24         <coordinates>102,0 103,1 104,0 105,1</coordinates>
25       </LineString>
26     </Placemark>
27   </Document>
28 </kml>
```

4.4.2. GeoJSON

GeoJSON ist ein Dateiformat für Geodaten, basierend auf der JSON Notation [HB+08a]. Dabei definiert GeoJSON im Kern zwei separate JSON Notationen. Geometry Object und Feature Object. Ein Geometry Object muss den Key "*type*" und den Key "*coordinates*" enthalten. Der Key "*type*" definiert die Art des Objekts, das heißt ob es sich um einen Punkt, Linie oder Polygon ("Point", "LineString", "Polygon") handelt. Davon abhängig ist auch die Formatierung des Keys "*coordinates*". In diesem sind die Koordinaten des Objekts enthalten. Zum Beispiel wären das bei einem geografischen Punkt die Breiten- und Längengrade.
Ein Feature Object muss dagegen zwei andere Keys enthalten. Der Key "*geometry*" muss ein Geometry Object als Wert enthalten. Der Wert für den Key "*properties*" kann dagegen ein beliebiges JSON sein. Das Feature Object stellt einen Art Wrapper für Geometry Objects dar. Durch den Key "*properties*" gibt es die Möglichkeit eigene Werte und Eigenschaften an einen geometrischen Punkt zu knüpfen.
In GeoJSON gibt es noch zwei weitere Objekttypen. *FeatureCollection* und *GeometryCollection*. Beide enthalten einen Key dessen Wert ein Array aus Feature bzw. Geometry Objects sein muss [HB+08b].

Ein Beispiel für ein GeoJSON [HB+08b]:

```
{ "type": "FeatureCollection",
  "features": [
    { "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [102.0, 0.5]},
      "properties": {"prop": "value"}
      },
    { "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
          ]
        },
      "properties": {
        "propone": "value",
        "proptwo": 0.0
        }
      },
    { "type": "Feature",
       "geometry": {
         "type": "Polygon",
         "coordinates": [
```

```
23          [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
24            [100.0, 1.0], [100.0, 0.0] ]
25          ]
26       },
27       "properties": {
28        "propzero": "value",
29        "propone": {"this": "that"}
30        }
31       }
32     ]
33   }
```

In diesem Codebeispiel sieht man ein FeatureCollections Object, welches drei Feature Objects enthält. Jedes Feature enthält jeweils ein anderes Geometry Object. Jedes Feature Object hat beispielhafte Eigenschaften unter "*properties*" angegeben. Man sieht auch pro Objekt eine eigene Koordinaten Angaben. Während es beim Punkt ein Array mit 2 Werten ist, hat der Polygon fünf Koordinaten in einem Array besitzt.

4.4.3. Fazit

Im Hinblick auf die White-Spot-Analyse und den dazugehörigen Anforderungen (Kapitel 2) dieser Arbeit, lässt sich ein Fazit bezüglich der beiden Datenformate KML und GeoJSON ziehen. KML bringt eine detaillierte Dokumentation [Goo16c] und hohe Flexibilität mit sich. Während GeoJSON sich mit einer Kompaktheit und einem simpleren Aufbau hervorhebt. In Anbetracht der Anforderung in Kapitel 2.4 wird das GeoJSON Format bevorzugt. Für die Verarbeitung von Daten in einer Weboberfläche und Nutzung von OpenStreetMaps wird JavaScript verwendet. GeoJSON bietet durch seine JSON Notation eine automatische Integration in JavaScript. Weiterhin wird GeoJSON von Frameworks wie Leaflet (Kapitel 5.5.1) bereits nativ unterstützt.

5. Softwareentwurf & Implementierung

Vor der Implementierung einer Software steht der Softwareentwurf an. Im Entwurf wird die Struktur und Funktionsweise der Software festgelegt. Es werden sowohl die Softwarekomponenten beschrieben, als auch extern verwendete Frameworks. Die Softwarekomponente trägt den Namen WASC, das für '**W**hitespot-**A**nalysis **S**oftware **C**omponent' steht und wird in Java entwickelt. WASC kann Daten im GeoJSON Format einlesen und nach Angabe von Konfigurationsparameter eine von drei verfügbaren Ausgaben erzeugen. Die Ausgaben sind komprimierte HTML und JavaScript Strings, welche die Ergebnisse einer White-Spot-Analyse im Browser des Nutzers visualisieren.

5.1. Komponentendiagramm

In Abbildung 5.1 sieht man grafisch die Komponenten von WASC dargestellt. WASC ist eine Erweiterungskomponente für bereits lauffähige Webapplikation basierend auf einem Java Backend. Somit ist WASC allein nicht lauffähig. WASC ist in zwei Hauptkomponenten zerlegt. Grundlage für die Aufteilung bieten die Anforderungen in Kapitel 2.3 und 2.4. Die 'Core' und 'Web' Komponente repräsentieren dabei die Aufteilung in Programm- und Nutzerschnittstelle. Die 'Core' Komponente enthält die Ressourcen, welche für die Funktionsweise von WASC essentiell sind. Die Ressourcen enthalten vor allem JavaScript Algorithmen, welche die White-Spot-Analyse durchführen. Weiterhin sind im 'Core' auch alle Entitäten zu finden, welche zur Betreibung der Rest-Schnittstelle nötig wären. Das enthaltene Datenmodell wird dabei für Umwandlung von SQL Ergebnissen genutzt und soll die Anforderung in Kapitel 2.2 abdecken. 'Web' ist die Komponente, welche ein Frontend für WASC bereitstellt. 'Web' ist von 'Core' abhängig. Die 'Web' Komponente hat eine vollständig integrierte UI, welche in der eigenen Webapplikation genutzt werden kann. 'Web' soll die Integration von 'Core' in anderen Webanwendungen einfacher gestalten. Im Falle einer Weiterentwicklung, enthält 'Web' alle nötigen Komponenten für die Nutzung von Serverlets.

5.2. Schnittstellen

Die Schnittstellen einer Software sind Zugangsmöglichkeiten der Software. WASC bietet drei Schnittstellen, die jeweils an verschiedene Nutzer gerichtet sind.

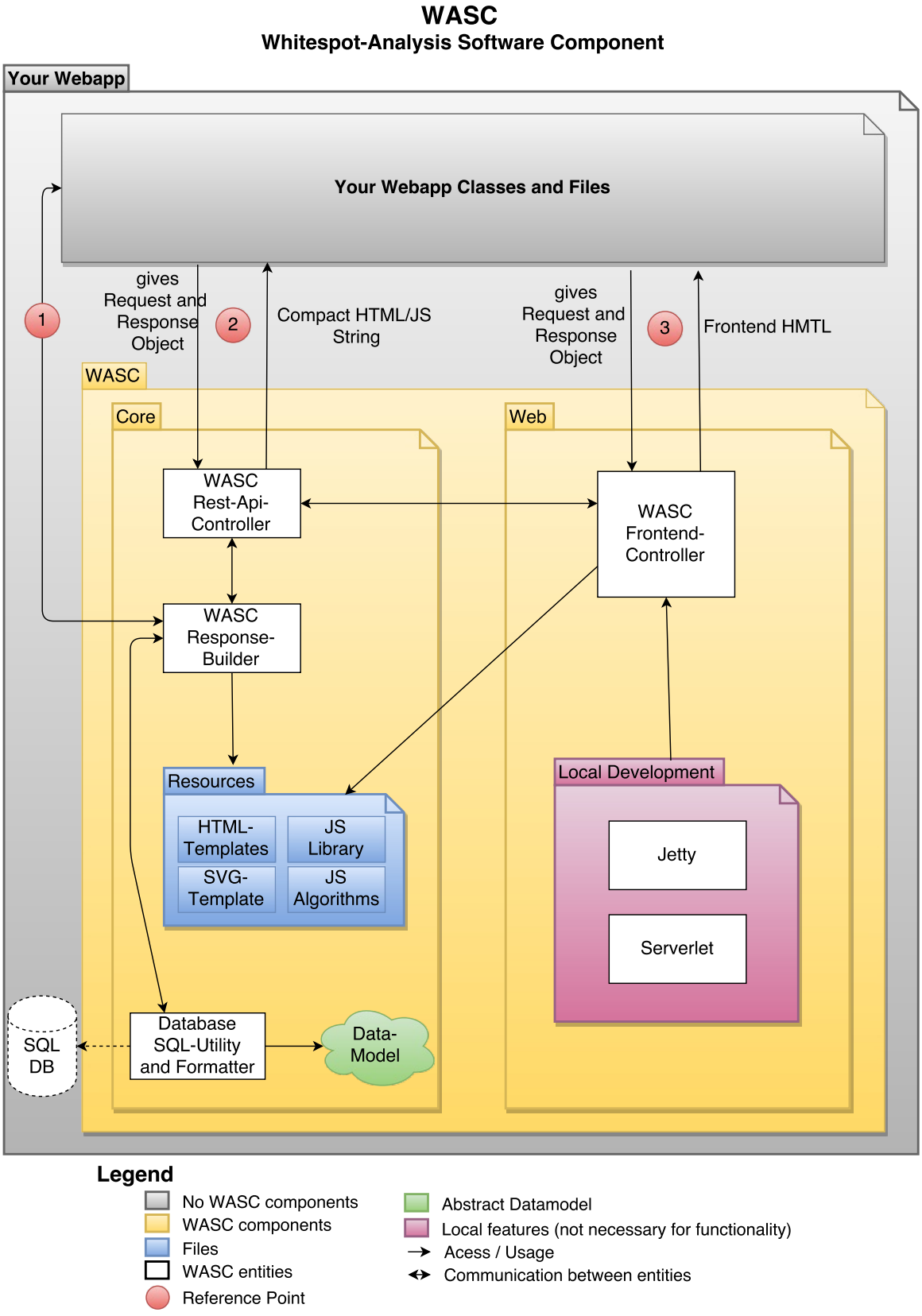

Abbildung 5.1.: Komponentendiagramm von WASC. Eigene Abbildung.

5.2.1. Programmierschnittstelle

Die Programmschnittstelle wird durch die Klasse *WASCRestResponseBuilder* repräsentiert. Damit lässt sich jeder Ausgabemöglichkeit von WASC als String zurückgeben. Außerdem hat man vollen Zugriff auf Konfigurationsparameter. Die Dateneingabe muss als GeoJSON String übergeben werden. Eine Überprüfung der Parameter wird nicht getätigt, da es sich hierbei um die Instanz handelt, welche die Eingaben den JavaScript Algorithmen übergibt. Die Programmierschnittstelle ist an Nutzer gerichtet, die nur reine WASC Funktionalität nutzen und vollständige Kontrolle über die Eingabeparameter wollen. Die Programmierschnittstelle ist im Komponentendiagramm (Abbildung 5.1) unter dem roten Referenzpunkt 1 zu finden.
Im Folgendem ist ein Codebeispiel für den Aufruf der Postleitzahlübersicht (Kapitel 5.4.3) zu sehen.

```
1 WASCRestResponseBuilder rrb = new WASCRestResponseBuilder(ressourcePath);
2 String response = rrb.buildResponseForPlzMap(geoJSON, plzKey, colorMin, colorMax,
      colorMid);
3 //Der String "response" enthaelt alle Daten und Algorithmen zur Darstellung im Browser.
```

5.2.2. REST-Schnittstelle

REST steht für **Re**presentational **S**tate **T**ransfer. REST ist ein Architektur-Stil für die Bereitstellung von Schnittstellen im World Wide Web [Daz12]. Eine REST-Schnittstelle ist dementsprechend eine Schnittstelle, die nach REST Prinzipien gestaltet ist.
WASC bietet eine REST-Schnittstelle, welche anderen Applikationen zur Verfügung gestellt werden kann. Dafür wird die Klasse *WASCRestController* bereitgestellt. Es wird lediglich ein *HttpServletRequest* Objekt und ein *HttpServletResponse* Objekt für den Vorgang benötigt. WASC überprüft dabei die Konfigurationsparameter und schreibt die nötigen Daten in die Response. Die REST-Schnittstelle richtet sich an Nutzer, die WASC Funktionalitäten in ihr eigenes Frontend einbauen möchten. Die REST-Schnittstelle ist im Komponentendiagramm (Abbildung 5.1) unter dem roten Referenzpunkt 2 markiert.
Eine vollständige Dokumentation der REST-API ist im Swagger Standard beschrieben und im Anhang A A.2 aufzufinden. Eine Darstellung der Swagger-UI in WASC bietet die Abbildung 5.8. Die REST-Schnittstelle richtet sich nach den drei Ausgabemöglichkeiten (Kapitel 5.4) von WASC. Die spezifischen Parameter pro Ausgabe sind in dem jeweiligen Kapitel beschrieben (Kapitel 5.4.1, Kapitel 5.4.2, Kapitel 5.4.3). Nicht Ausgabe-spezifische Parameter werden im Folgenden erklärt.

wasc_api Definiert welche Ausgabemöglichkeit verlangt wird. Es werden drei verschiedene Werte angenommen, die jeweils eine Ausgabemöglichkeit repräsentieren. Abdeckungsübersicht = 'circlemap', Intensitätsübersicht = 'heatmap', Postleitzahlübersicht = 'plzmap'.

data Enthält alle Geodaten im GeoJSON Format.

query	Enthält eine komplette SQL-Query, welche Geodaten als Ergebnis liefert.
lat_field	Der Parametername für den Breitengrad eines Datenpunktes. Der Parametername repräsentiert einen field key des Datenbankergebnisses.
lng_field	Der Parametername für den Längengrad eines Datenpunktes. Der Parametername repräsentiert einen field key des Datenbankergebnisses.

Alle **fett** gedruckten Parameternamen sind Pflichtangaben. Die unterstrichenen Parameter stehen in einer 'entweder oder' Beziehung. Es muss entweder der 'data' Parameter vorliegen oder der 'query' Parameter. Wenn der 'query' Parameter vorliegt, so müssen auch die *kursiven* Parameter 'lat_field' und 'lng_field' vorliegen.

5.2.3. Frontendschnittstelle

Die Frontendschnittstelle von WASC ist eine komplett grafische Benutzeroberfläche, welche über drei Menüpunkte alle WASC Funktionalitäten integriert. Unter dem Menüpunkt 'Analysis' lässt sich eine White-Spot-Analyse mit der Intensitätsübersicht oder Abdeckungsübersicht durchführen. Der Menüpunkt 'Postal Code Germany' erlaubt die Nutzung der Postleitzahlübersicht. In 'Analysis' und 'Postal Code Germany' lassen sich Geodaten auf drei Arten einlesen. Durch Angabe einer SQL Query, durch Eingabe von einem GeoJSON String oder durch das Hochladen einer Textdatei, in der ein GeoJSON enthalten ist. Weiterhin lassen sich auch alle Konfigurationsparameter (Kapitel 5.4) durch grafische HTML Elemente einstellen. Unter dem Menüpunkt 'Rest-Api' befindet die Integration der Swagger-UI (Kapitel 5.5.2). Dort lässt sich die REST-Schnittstelle von WASC ausprobieren und ein Download für die Swagger Konfiguration finden. Die Steuerung der Schnittstelle übernimmt die Klasse *WASCFrontendController* unter Zuhilfenahme von einem *HttpServletRequest* Objekt und einem *HttpServletResponse* Objekt. Jegliche Interaktion mit dem Frontend übernimmt WASC automatisch. Die Frontendschnittstelle ist an Nutzer gerichtet, die den kompletten Umfang von WASC nutzen wollen ohne viele Integrationsschritte zu brauchen. Eine Aufnahme vom Menüpunkt 'Analysis' der Frontendschnittstelle ist in der Abbildung 5.2 zu sehen. Die Frontendschnittstelle ist im Komponentendiagramm (Abbildung 5.1) unter dem roten Referenzpunkt 3 zu finden.

5.3. Dateneingabe

Datensätzen müssen in einer strukturierten Form vorliegen, damit diese von einer Software gelesen und verarbeitet werden können. WASC enthält drei Möglichkeiten zur Dateneingabe. Eine direkte Dateneingabe durch GeoJSON, durch Abfrage einer SQL-Datenbank oder durch Nutzung des internen Datenmodells. Die beiden letzten genannten Möglichkeiten werden vor der Verarbeitung in GeoJSON umgewandelt. Eine Veranschaulichung der Funktionsweise aller drei Eingabemöglichkeiten ist in Abbildung 5.3 zu sehen.

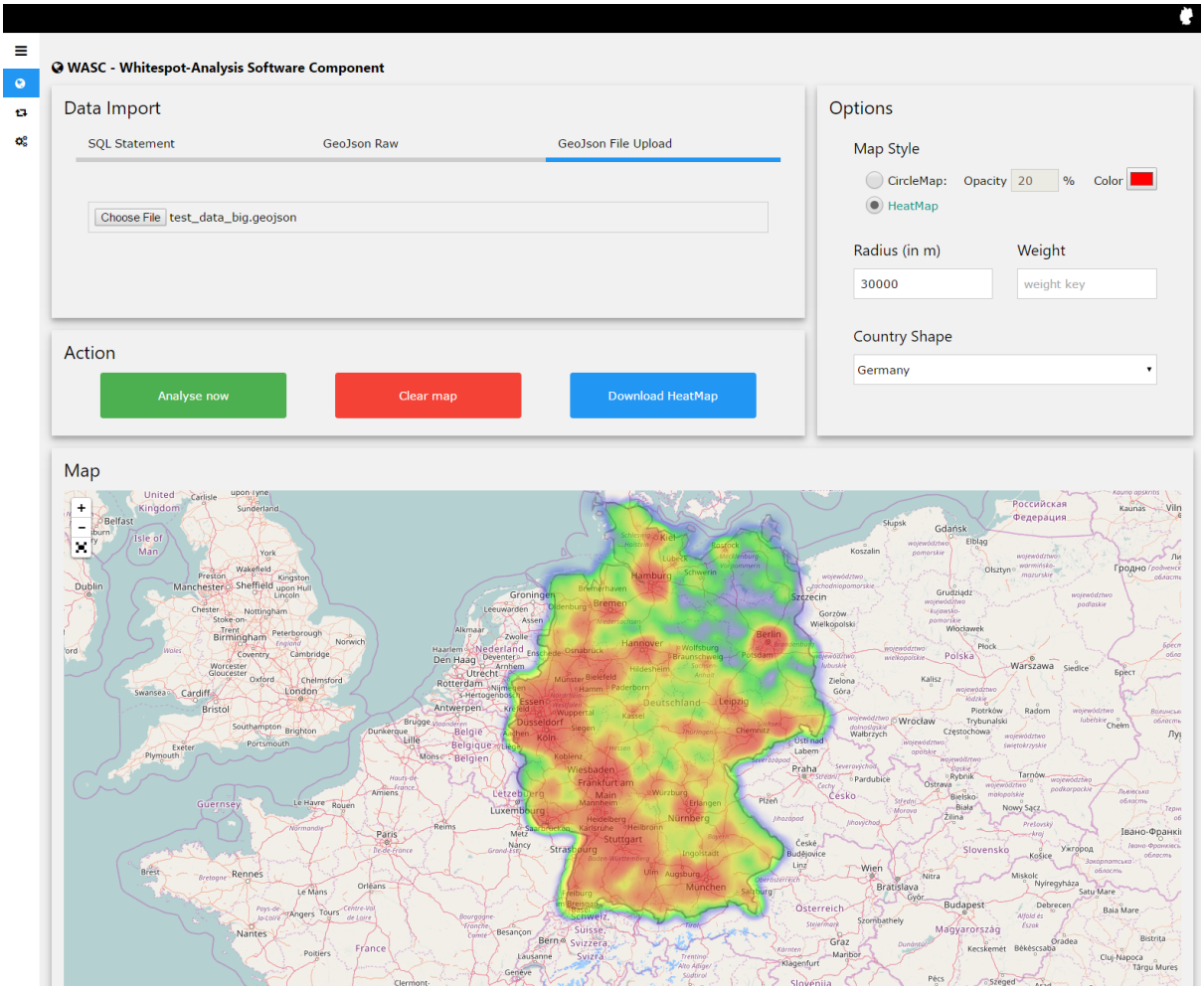

Abbildung 5.2.: Momentaufnahme von der WASC Frontendschnittstelle. Abgebildet ist der Inhalt des 'Analysis' Menüpunktes, in dem ein Testdatensatz hochgeladen wurde. Es wurde die Intensitätsübersicht als Ausgabe gewählt. Die anderen Menüpunkte sind als minimierte Symbole oben links im Bild zu sehen. Eigene Abbildung.

5.3.1. GeoJSON

In WASC wird die direkte Dateneingabe über das GeoJSON Format (Kapitel 4.4.2) getätigt. Dabei stellt der gesamte Datensatz eine FeatureCollection dar. Jedes Feature dieser FeatureCollection enthält dabei nur einen Punkt als geometrisches Objekt. Andere geometrische Objekte werden von WASC nicht unterstützt. Jedes Feature darf eine beliebige Anzahl an Properties aufweisen. Konfigurationsparameter (Kapitel 5.4) pro Punkt sind Bestandteil der Propeties vom einzelnen Feature. Der Datensatz kann in Form eines Strings in der REST-API, in der Programmierschnittstelle, als auch in der Frontendschnittstelle genutzt werden. Weiterhin

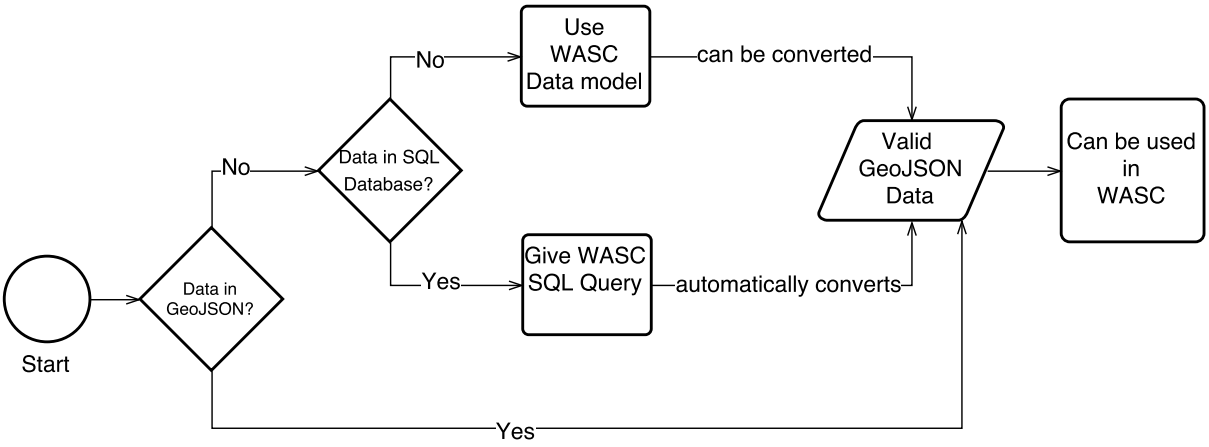

Abbildung 5.3.: Ablaufdiagramm für einen Datenimport in WASC. Eigene Abbildung.

lassen sich in der Frontendschnittstelle auch Textdateien importieren. Dabei darf eine Textdatei nur einen Datensatz in GeoJSON Form enthalten.

5.3.2. SQL Query

Eine weitere Möglichkeit der Dateneingabe ist eine SQL Query. Dafür benötigt WASC eine gültige Datenbankanbindung. Die Datenbankanbindung muss vom Java Interface *DataSource* erben. Die SQL Felder für Breiten- und Längengrad eines geografischen Punktes müssen mitangegeben werden. Die Dateneingabe über die SQL Query lässt sich in der REST-API, als auch in der Programmierschnittstelle nutzen. Nach der Datenbankabfrage wandelt WASC die Ergebnistabelle automatisch in ein GeoJSON um. Dabei stellt jedes SQL Feld einen Property Eintrag im Feature dar. Ausnahme bilden hier die jeweiligen Breiten- und Längengrade Felder, welche dem geografischen Objekt zugeordnet werden.

5.3.3. Datenmodell

Das Datenmodell von WASC ist primär als alternative Dateneingabe gedacht. Damit soll die Dateneingabe erleichtern werden, wenn Daten nicht in einer Datenbank vorliegen oder nur als Objekte im Speicher sich aufhalten. Das Datenmodell, welches aus zwei Klassen besteht baut stark auf dem GeoJSON Format auf. Dabei spiegelt *WASCDatensatz* eine *FeatureCollection* und der *WASCDatenPunkt* ein *Feature* inklusive des geografischen Objektes. Ein *WASCDatensatz* Objekt enthält eine Liste aus *WASCDatenPunkt* Objekten. Ein *WASCDatenPunkt* Objekt besteht aus zwei double Werten für den Breiten- und Längengrad und einer beliebig großen Key-Value Liste. *WASCDatensatz* kann durch die Methode *convertToGeoJson()* jederzeit in das GeoJSON Format überführt werden. Damit lassen sich die Daten an jeder Schnittstelle von WASC nutzen.

5.4. Ausgabe & Konfigurationsparameter

In diesem Abschnitt werden die Ausgaben und deren Konfigurationsparameter von WASC beschrieben. WASC bietet drei Ausgabemöglichkeiten. Eine Abdeckungsübersicht, eine Intensitätsübersicht und eine Abdeckung pro Postleitzahlbereich in Deutschland. Jede Ausgabe hat zusammenhängende, als auch eigenen Konfigurationsparameter. Alle Ausgaben von WASC sind komprimierte HTML und JavaScript, somit ist für die Darstellung der Ergebnisse ein Browser nötig.

5.4.1. Abdeckungsübersicht

Das primäre Ziel dieser Ausgabe, ist es eine Übersicht über die Abdeckung des gegebenen Datensatzes zu visualisieren. Damit lässt sich die Verteilung der jeweiligen Datenpunkte feststellen, als auch Lücken im Datensatz finden. Jeder Datenpunkt mit einem gültigen Längen- und Breitengrad wird dabei als Kreis auf die OSM Karte[1] gezeichnet. Jeder Kreis lässt sich anklicken um zusätzliche Informationen zu dem Datenpunkt anzeigen zu lassen. Die Informationen sind in zwei Kategorien aufgeteilt. 'Properties' und 'Info'. Properties zeigt alle Key-Value Eigenschaften, die der Datenpunkt enthält. Das sind die gleichen Wertpaare, die im GeoJSON Format (Kapitel 4.4.2) bei einem Feature unter 'properties' zu finden sind. Info enthält dabei den Längen- und Breitengrad des Datenpunktes.
Die Abdeckungsübersicht enthält folgende Konfigurationsparameter, welche auch bei der REST-Schnittstelle (Kapitel 5.2.2) genutzt werden:

radius	Definiert den gezeichneten Radius pro Datenpunkt in Meter. Ein Wert von 30000 entspricht 30km Radius. Es kann auch ein Property Key eingegeben werden um den Radius pro Datenpunkt individuell zu setzen.
opacity	Definiert die Deckkraft eines gezeichneten Kreises in Prozent. 80 entspricht einer Deckkraft von 80%. Standardwert ist 20.
circlecolor	HEX Farbcode um die Farbe der gezeichneten Kreise zu ändern. #000000 entspricht der Farbe Schwarz. Standardwert ist #FF0000 (Rot).
countryshape	Markiert die Fläche des angegebenen Landes in leicht grauem Ton mit den Landesgrenzen. Der Landname ist zusammenhängend in Englischer Sprache anzugeben. Standardwert ist 'Germany'.

Alle **fett** gedruckten Parameternamen sind Pflichtangaben, die restlichen Parameter sind optional.

[1] https://www.openstreetmap.org/

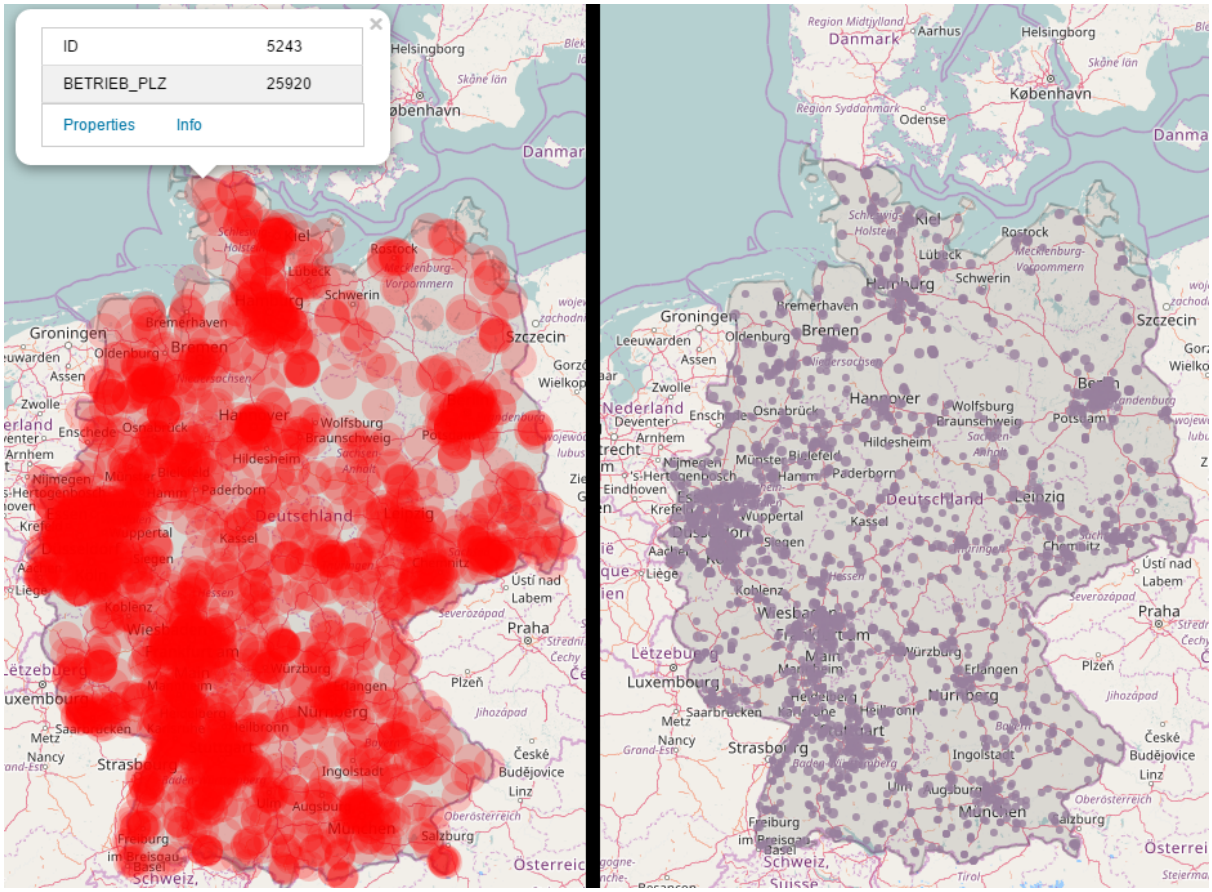

Abbildung 5.4.: Zwei Abbildungen der Abdeckungsübersicht. Eigene Abbildung.

In der Abbildungen 5.4 sieht man zwei Abbildungen der Abdeckungsübersicht. In der linken Abbildung wurde ein Radius von 20km gewählt und die Deckkraft auf 20% gesetzt. Außerdem wurde ein Datenpunkt ausgewählt um seine Eigenschaften anzeigen zu lassen. In der rechten Abbildung wurde die Farbe der Kreise auf lila gesetzt und der Radius pro Datenpunkt auf 5km festgelegt, außerdem wurde die Deckkraft auf 80% erhöht. Es wurde bei beiden Abbildungen der gleiche Testdatensatz verwendet.

5.4.2. Intensitätsübersicht

Die Intensitätsübersicht soll einen Überblick darüber geben, in welchen Bereichen sich die Datenpunkte ballen. Das Ganze wird in Form einer Heatmap repräsentiert. Die Heatmap hat vier markante Farbbereiche, welche mit dem Umkreis der Datenpunkte zusammenhängen. Der Umkreis wird durch den Radius bestimmt. Befindet sich ein Datenpunkt A im Umkreis eines anderen Datenpunktes B und Datenpunkt A ist maximal 0%- 15% des Radiuswertes vom Datenpunkt B entfernt, so wird dieser Bereich rot gefärbt. Befindet sich Datenpunkt B maximal 15%- 45% entfernt vom Radiuswert, so wird der Bereich gelb gefärbt. Bei 45%- 75% wird die

Farbe Grün gewählt und bei 75%- 100% die Farbe Blau. Wenn die Datenpunkte sich mit ihrem Umkreis nicht berühren, so wird pro Datenpunkt ein Bereich in der Größe des Umkreises gezeichnet. Dieser Bereich ist in sich nach den vier Farben und der jeweiligen Prozentangabe gefärbt.
Die Intensitätsübersicht eignet sich vor allem für die Feststellung von Ballungsräumen in einem Datensatz. Die Konfiguration wird mit folgenden Parametern eingestellt, diese werden auch bei der REST-Schnittstelle (Kapitel 5.2.2) genutzt:

radius	Definiert den gezeichneten Radius pro Datenpunkt in Meter. Ein Wert von 30000 entspricht 30km Radius. Es kann auch ein Property Key eingegeben werden um den Radius pro Datenpunkt individuell zu setzen.
weight	Definiert die Gewichtung eines Datenpunktes im Vergleich zu anderen Datenpunkten. Hat Datenpunkt A ein Gewicht von 1 und Datenpunkt B ein Gewicht von 10, so wird der Radius von A auf 1/10 gesetzt. Die Gewichtung wird durch einen Property Key gesetzt. Standardgewichtung ist 1 für jeden Datenpunkt.
countryshape	Markiert die Fläche des angegebenen Landes in leicht grauem Ton mit den Landesgrenzen. Der Landname ist zusammenhängend in Englischer Sprache anzugeben. Standardwert ist 'Germany'.

Alle **fett** gedruckten Parameternamen sind Pflichtangaben, die restlichen Parameter sind optional.

In der Abbildungen 5.5 sieht man zwei Abbildungen der Intensitätsübersicht. In beiden Abbildungen wurde ein Radius von 30km gewählt und der gleiche Testdatensatz genutzt. In der linken Abbildung wurde eine Gewichtung von 1 für alle Datenpunkte gewählt. In der rechten Abbildung wurde die Gewichtung aus der ersten Ziffer der Postleitzahl entnommen und wie folgt verrechnet ((Ziffer - 9) * (-1)). Somit werden alle Postleitzahlbereiche mit einer großen ersten Ziffer weniger gewichtet als Postleitzahlbereiche mit einer kleinen ersten Ziffer. Dies spiegelt sich auch in der Abbildung. Bayern ist stark geschrumpft und auch Stuttgart hat abgenommen. Dafür hat die Gegend um Leipzig stark zugenommen.

5.4.3. Postleitzahlübersicht

Die Postleitzahlübersicht ermöglicht eine Darstellung über die Verteilung von Datenpunkten pro Postleitzahlbereich in Deutschland. Die Darstellung erfolgt mithilfe einer SVG, die alle Postleitzahlbereiche in Deutschland darstellt. Zunächst wird der Datensatz nach Postleitzahlen durchsucht. Zu jeder Postleitzahl werden die entsprechenden Datenpunkte zugeordnet. Die Postleitzahl mit den meisten zugeordneten Datenpunkten stellt das Maximum dar. Die Postleitzahl mit den wenigstens Datenpunkten, das Minimum. Anschließend wird eine Farbe dem Minimum und eine andere Farbe dem Maximum zugeordnet. Mit einer Berechnung eines

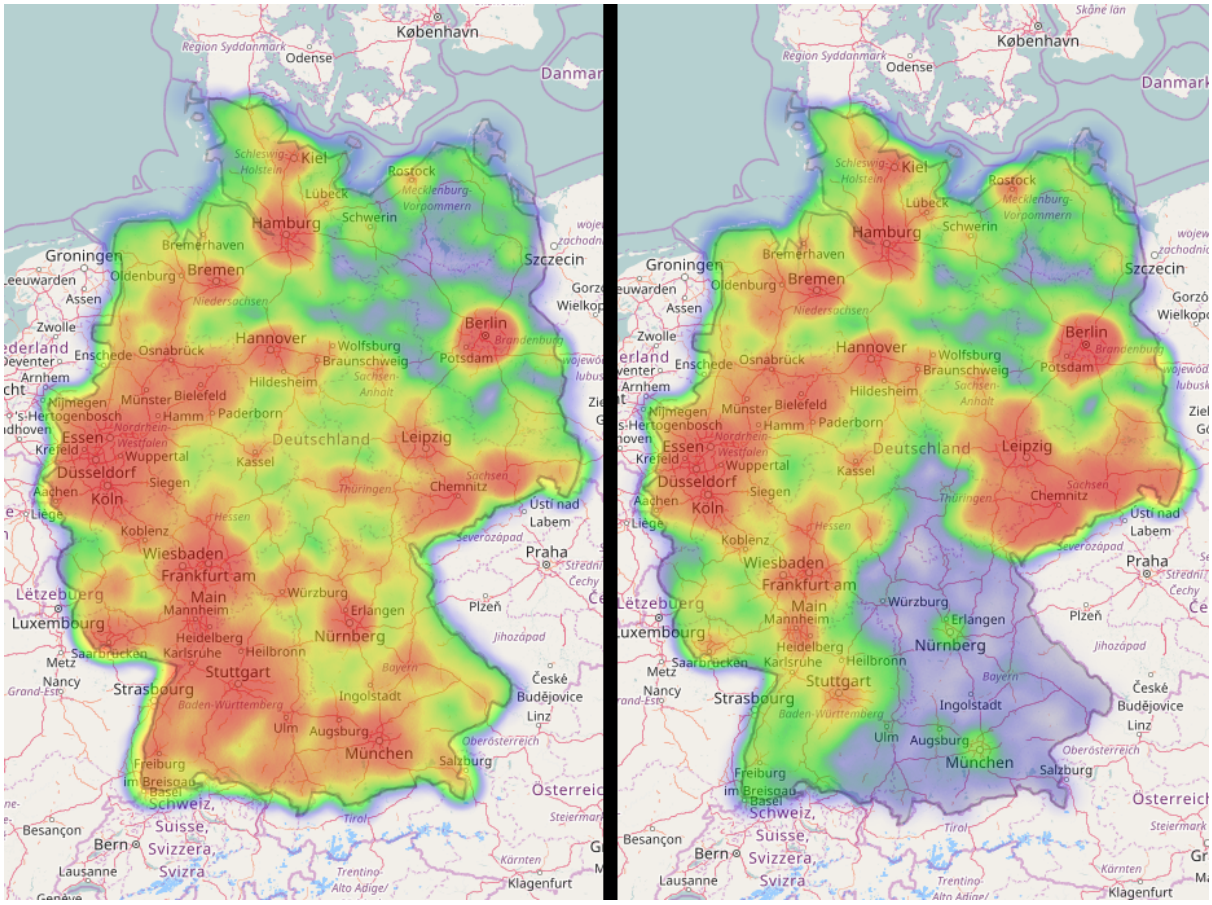

Abbildung 5.5.: Zwei Abbildungen der Intensitätsübersicht. Eigene Abbildung.

Farbverlaufs der zwei Farben, lässt sich jede weitere Anzahl der Datenpunkte zwischen Minimum und Maximum darstellen. Jeder Postleitzahlbereich vom SVG wird dementsprechend gefärbt. Somit werden Postleitzahlbereiche mit vielen Datenpunkten in einer ähnlichen Farbe die das Maximum besitzt dargestellt. Bereiche mit wenigen Datenpunkten erhalten eine Farbe ähnlich dem Minimum.
Die Postleitzahlübersicht enthält folgende Konfigurationsparameter, welche auch bei der REST-Schnittstelle (Kapitel 5.2.2) genutzt werden:

plzkey	Angabe des Property Key, welcher die Postleitzahl des Datenpunktes enthält.
colormin	HEX Farcode des Minimums. Standardwert ist #b3c6ff (Hellblau).
colormax	HEX Farcode des Maximums. Standardwert ist #002db3 (Dunkelblau).
colormid	Optionale HEX Farcode für Werte zwischen Minimum und Maximum.

colorzero — Dieser Parameter ist nicht Teil der REST-Schnittstelle und ist nur in der Frontendschnittstelle zu finden. Optionale HEX Farcode für Postleitzahlbereiche mit 0 Datenpunkten. Nur in der Frontendschnittstelle verfügbar. Standardwert ist #FFFFFF (Weiß).

Alle **fett** gedruckten Parameternamen sind Pflichtangaben, die restlichen Parameter sind optional.

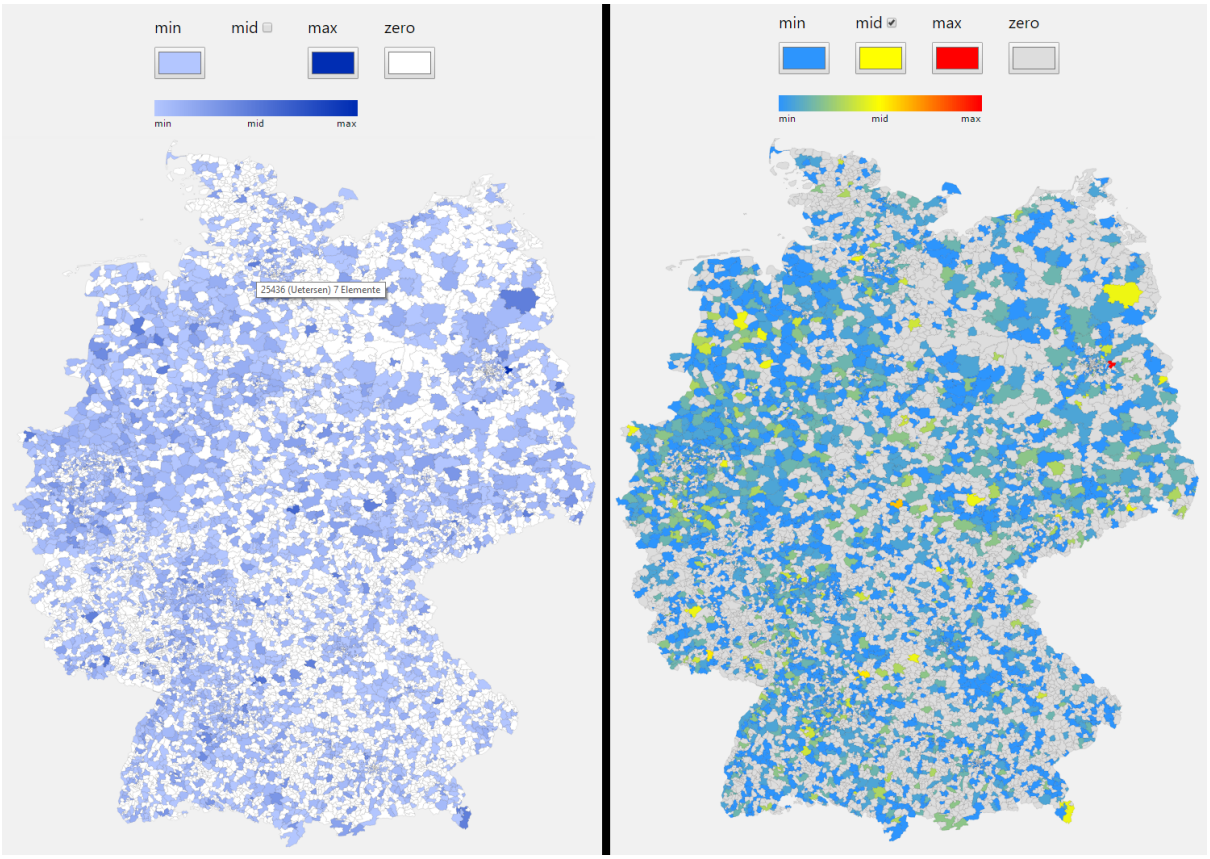

Abbildung 5.6.: Zwei Abbildungen der Postleitzahlübersicht. Eigene Abbildung.

In der Abbildungen 5.6 sieht man zwei Abbildungen der Postleitzahlübersicht. In der linken Abbildung sieht man die Standardkonfiguration der Farben. Außerdem wurde mit dem Mauszeiger über einen Postleitzahlbereich gefahren und es erscheint ein Pop-up, welches die Postleitzahl, den Ort, sowie die Anzahl der zugeordneten Datenpunkte enthält. In der rechten Abbildung sieht man eine individuelle Konfiguration der Farben. Hier wurden Bereiche mit einer großen Anzahl an Datenpunkte rot gefärbt. In diesen Bereich fällt z.B. eine Postleitzahl in Berlin. Bereiche mit mittlerer Abdeckung werden gelb gefärbt. Übergänge zwischen den Farben sind auch zu sehen. Man sieht einige grüne Bereiche, welche den Übergang zwischen mittlerer und minimaler Abdeckung darstellen. Für beide Abbildungen wurde der gleiche Testdatensatz verwendet.

5.5. Frameworks

Die Entwicklung einer Software kann je nach Aufgabestellung einen immensen Aufwand benötigen. Um den Aufwand zu verkleinern und den Code strukturiert zu halten werden Frameworks bzw. Bibliotheken verwendet. Bei einer White-Spot-Analyse, sind vor allem Technologien gefragt, die mit geografischen Geodaten arbeiten können. Außerdem können grafische Frameworks das Nutzererlebnis verbessern. In diesem Kapitel werden drei Frameworks angeschaut, welche die Entwicklung der Softwarekomponente erleichtern.

5.5.1. Leaflet.js

Leaflet [Aga15a] ist ein Open-Source JavaScript Framework zur Erstellung und Nutzung von interaktiven Landkarten. Leaflet ermöglicht grundlegende Markierungsmöglichkeiten auf Karten, wie z.B. das Setzen von Punkte oder Flächen. Leaflet versucht dabei möglichst leichtgewichtig zu bleiben, weswegen auf ein reichhaltiges Feature Angebot verzichtet wird. Zusätzliche Features können durch ein Pluginsystem von Drittentwicklern hinzugefügt werden. Darunter finden sich Möglichkeiten wie Geodcoding oder auch Distanzmessung, die als Plugins integriert werden [Aga15c]. Leaflet lässt sich sowohl mit OpenStreetMaps als auch mit Google Maps nutzen. Weiterhin unterstützt Leaflet von Haus aus den Geodaten Standard GeoJSON (Kapitel 4.4.2).

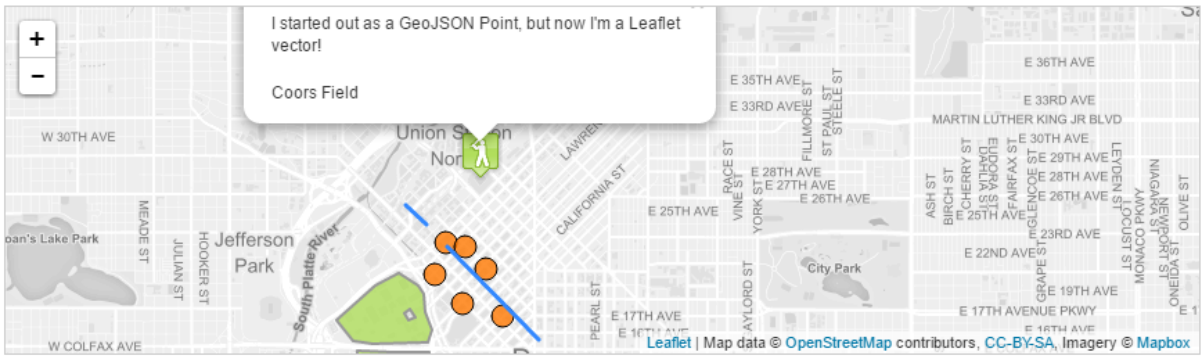

Abbildung 5.7.: Mögliche Darstellung eines GeoJSON mit Leaflet. In der Abbildung sieht man die Visualisierung eines GeoJSON, welcher einige Kreise, Linien und Flächen darstellt. Außerdem sieht man einen Tooltip mit Informationen. Die Abbildung ist eine Aufnahme aus dem Tutorial von Leaflet [Aga15b].

5.5.2. Swagger und Swagger-UI

Swagger [Swa16b] ist eine formale Spezifikation für REST Schnittstellen. Mit Swagger kann eine REST Schnittstelle dokumentiert werden. Dort wird klar definiert wie man eine bestimme

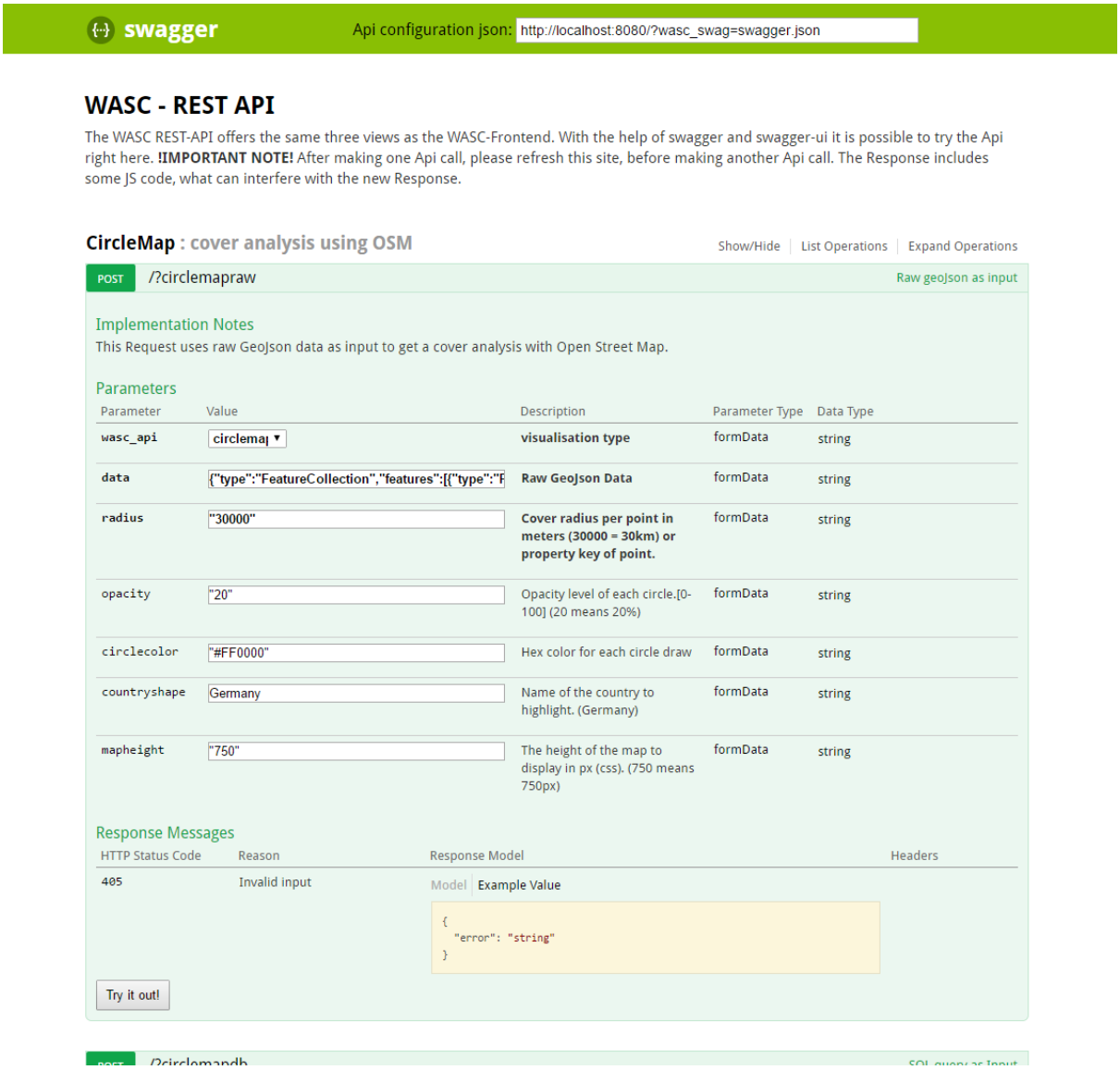

Abbildung 5.8.: Screenshot aus der Swagger-UI von WASC. Eigene Abbildung.

REST Schnittstelle ansprechen muss und was man als Antwort erhält. Swagger wird außerdem von viele anderen Projekte unterstützt. Dazu gehört z.B Swagger-UI [Swa16a]. Swagger-UI erzeugt automatisch ein Frontend aus einer Swagger Spezifikation. Dadurch lassen sich REST Schnittstellen sofort zum Testen nutzen. Alle wichtigen Einstellungen werden angezeigt. Eingabeparameter lassen sich in der UI ändern und Testrequest abschicken. In der Abbildung 5.8 sieht man beispielhaft die REST-Schnittstelle von WASC. Die Swagger Spezifikation zu der jeweiligen REST-API ist durch den im Header angezeigten Link erreichbar.

5.5.3. GeoTools

GeoTools [geo16] ist ein Open-Source Java Framework für jegliche Arbeit mit Geospatial Data (Kapitel 3.3). GeoTools setzt sich aus vielen verschiedenen Paketen zusammen, welche einzeln genutzt werden können. Jedes Paket implementiert eine in sich geschlossenen Bibliothek. Beispielhaft sei hier das Paket 'gt-geojson' genannt, welches ein Datenmodell für GeoJson (Kapitel 4.4.2) mitbringt, als auch die dazugehörigen Umwandlungstools. GeoTools ist eines der Gründungsprojekte der Open Source Geospatial Foundation [2] und ist somit Open Source & Open Development.

5.6. Fazit - WASC

Bei der Entwicklung von WASC wurde vor allem darauf geachtet, dass Kernelemente sowohl modular und unabhängig, als auch State of the Art (aktueller Stand der Technik) sind. Deswegen wurde vor allem auf etablierte Frameworks wie z.B. GeoTools (Kapitel 5.5.3) gesetzt, als auch auf aktuelle Standards wie GeoJSON (Kapitel 4.4.2) und Swagger Spezifikation (Kapitel 5.5.2). Damit soll die Softwarekomponente für eventuelle Weiterentwicklungen vorbeireitet sein. Die Berechnungsalgorithmen sind in JavaScript geschrieben und sind somit an keine spezifisches Backend gebunden. Auf diese Weise lassen sich die Kernfunktionalität von WASC z.B. auch mit einem C++ oder Python Backend realisieren. Außerdem bleibt der Berechnungsaufwand clientseitig und entlastet damit den Server.

[2] http://www.osgeo.org/

6. Evaluation

Dieses Kapitel befasst sich mit der Evaluation von WASC anhand von drei Datensätzen. Bei den Datensätzen handelt es sich um geografische Datensätze (Geospatial Data), welche sich größtenteils auf Deutschland beschränken. Die ersten zwei Datensätze stammen vom Fraunhofer IAO. Dabei enthält der erste Datensatz Kfz-Werkstätten und der zweite Datensatz ist ein generiertes Ergebnis aus dem Abgleich zweier Datenbanken. Der dritte Datensatz [AG16c] [AG16b] stammt von der Rheinisch-Westfälischen Elektrizitätswerke (RWE AG) und enthält Ladesäulen für Elektroautos. Der Datenbestand und das Thema eMobility der RWE AG wird jedoch seit dem 01.04.16 von der Firma Innogy [inn16] verwaltet.

6.1. White-Spot-Analyse von Werkstätten

Der gegebene Datensatz enthält 681 Einträge, jeder Datenpunkt repräsentiert eine Werkstatt in Deutschland. Der Datensatz wird auf seine Verteilung in Deutschland untersucht. Der Datensatz wurde mit Hilfe einer SQL Query (Kapitel 5.3.2) in ein GeoJSON umgewandelt. Bei der Verteilung wird geprüft ob der Datenbestand eine vollständige Abdeckung in Deutschland gewährleistet. Eine vollständige Abdeckung ist gewährleistet, wenn für jeden Punkt in Deutschland in einem Umkreis von 30km, mindestens ein Datenpunkt des Datensatzes zuordnen lässt. Der Grund dafür ist, dass Kunden eine Werkstatt in der Region empfohlen werden soll, die maximal 30km entfernt ist. Dafür wird die Abdeckungsübersicht (Kapitel 5.4.1) von WASC genutzt. Die Analyse wird mit folgenden Konfigurationsparametern durchgeführt. Der Radius pro Datenpunkt wird auf 30000 Meter gesetzt, die Deckkraft wird auf 50% eingestellt und als Abdeckfarbe wird Rot (#FF0000) gewählt.

In der Abbildung 6.1 sieht man das Ergebnis der Analyse. Jeder Datenpunkt wurde inklusive seinem Umkreis als rote Kreisfläche markiert. Viele der Flächen überlappen sich stark und erhöhen damit die Deckkraft. Teilweise gibt es Stellen, die von keiner Kreisfläche berührt werden. Dies ist vor allem im nördliche Teil von Deutschland zu sehen. Aus der Analyse kann man folgende Schlussfolgerung herausziehen. Der gegebene Datensatz erfüllt nicht die vollständige Abdeckung in Deutschland, bei einem gegebenen Umkreis von 30km. Kleinere Lücken sind sowohl im Süden und der Mitte von Deutschland zu finden. Ein Beispiel wäre die Lücke zwischen Heilbronn und Nürnberg. Größere Lücken sind im nördlichen Teil von Deutschland zu finden. Zwischen Schwerin und Wolfsburg, als auch südlich von Oldenburg. Weiterhin lässt sich sagen, dass die Abdeckung in dicht besiedelten Gebieten wie Stuttgart, München, Köln, Berlin oder Hamburg ausreichend vorhanden ist.

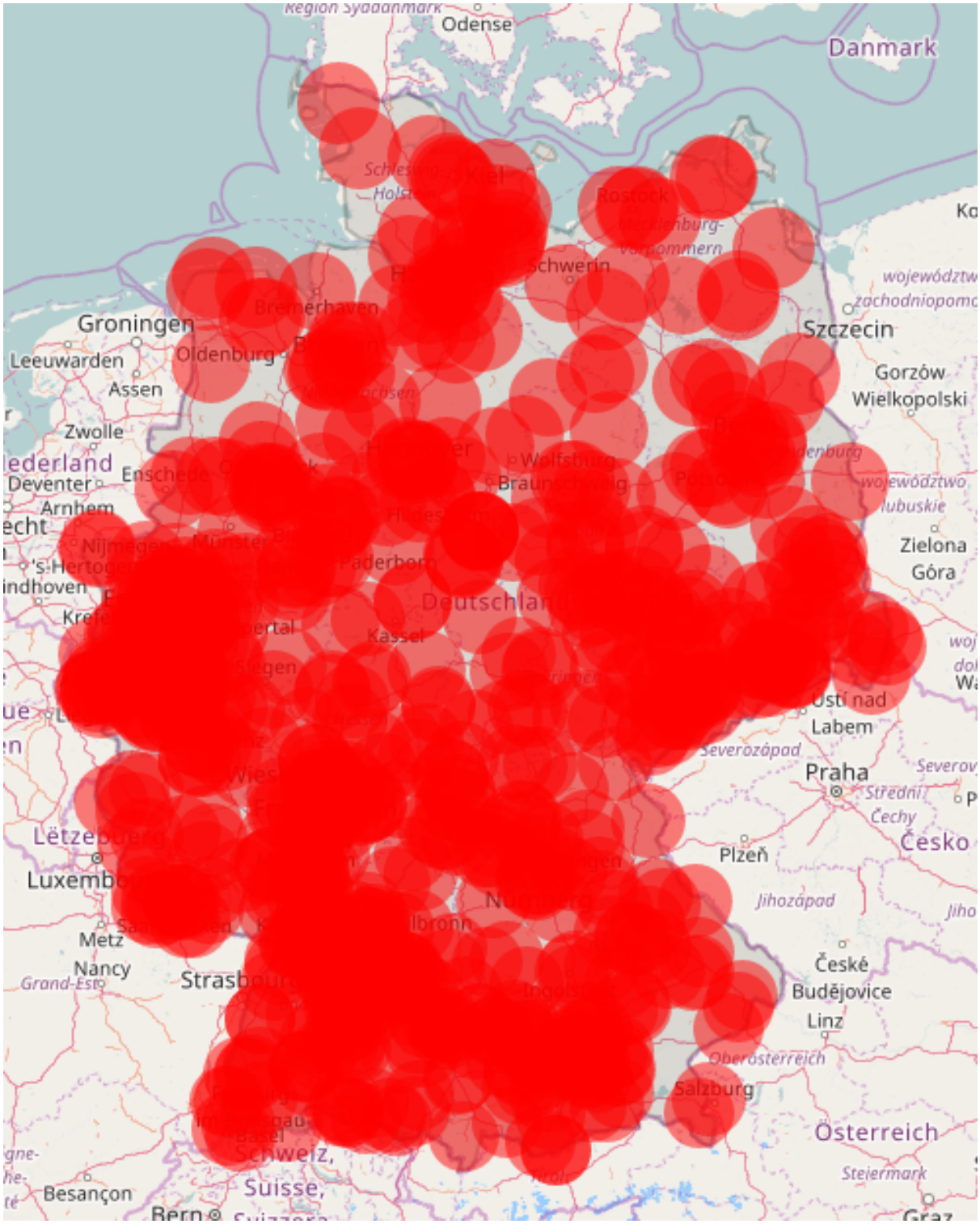

Abbildung 6.1.: Ergebnis der Abdeckungsübersicht von Werkstätten. Eigene Abbildung.

6.2. White-Spot-Analyse vom Datenbankabgleich

Bei dem folgenden Datensatz handelt es sich um einen generierten Datensatz aus dem Vergleich zweier Datenbanken. Der Datensatz gibt die Bereiche in Deutschland an, in denen keine geeignete Werkstätten für eine Kundenempfehlung innerhalb der Datenbank gefunden wurde. Der Datensatz ist in diesem Fall eine Ansammlung von White Spots. Dieser enthält 3073 Einträge. Der Datensatz wurde mit Hilfe einer SQL Query (Kapitel 5.3.2) in ein GeoJSON umgewandelt. Es wird untersucht in welchen Postleitzahlbereichen eine mittlere bis starke Verteilung der White Spots zu finden ist. Aus dem gewonnen Wissen soll ein Folgeprozess nach neuen Qualitätswerkstätten in den Bereichen der White-Spots suchen. Dieser Folgeprozess ist nicht Teil dieser Analyse. Dafür wird die Postleitzahlübersicht (Kapitel 5.4.3) von WASC genutzt. Die Analyse wird mit folgenden Konfigurationsparametern durchgeführt. Die Farbe für minimale Anzahl an Elementen in einem Postleitzahlbereich wird auf Weiß (#FFFFFF) gesetzt. Der Bereich für mittlere und maximale Anzahl wird auf Blau (#0000FF) gesetzt. Dadurch entsteht eine stärkere farbliche Betonung des mittleren und maximalen Bereichs.

In der Abbildung 6.2 sieht man die Deutschlandkarte unterteilt in die Postleitzahlbereiche. Einige Bereiche sind stark Blau gefärbt, andere verblassen ins Weiße. Jeder stark blaue Bereich ist von hellblauen Bereichen umgeben. Der größte Teil der Abbildung ist jedoch weiß gefärbt. Aus der Abbildung lassen sich einige Folgerungen herauslesen. Eine starke Verteilung von White Spots findet sich in 3 Gebieten. Nördlich von Brandenburg, bei Oldenburg und Osnabrück, aber auch im mittlere Bereich Deutschlands bei Kassel bis Thüringen. Weiterhin findet sich eine mittlere Anhäufung in Rheinland-Pfalz, südlich von Nürnberg und östlich von München bei Salzburg. Daraus lässt sich ein leichtes Muster erkennen, welches eine Verteilung der White-Spots an den Grenzen von Deutschland aufweist, mit den Ausnahmen bei Kassel und Nürnberg. Bereiche um Großstädte wie Köln, Stuttgart oder Hamburg enthalten keine White Spots.
Aus der Analyse lässt sich folgende Konsequenz ziehen. WASC hat aufgezeigt, dass vor allem im nordöstlichen und nordwestlichen Bereich mehr Qualitätswerkstätte in die Datenbank aufgenommen werden müssen. Bereiche um Metropolen sind dagegen ausreichend abgedeckt.

6.3. White-Spot-Analyse von Ladesäulen

Der vorliegende Datensatz enthält 1487 Einträge [AG16b] [AG16c]. Der Datensatz wurde Mithilfe des WASC Datenmodells (Kapitel 5.3.3) in ein GeoJSON umgewandelt. Jeder Eintrag repräsentiert eine Ladesäule für Elektroautos. Hierbei handelt es sich nicht um alle Ladesäulen in Deutschland, sondern nur um Ladesäulen die mit dem Rheinisch-Westfälischen Elektrizitätswerke in Verbindung stehen. Es wird untersucht wie die Verteilung der Ladesäulen in Deutschland ist. Außerdem wird geprüft ob Großstädte ausreichend abgedeckt sind. Dafür

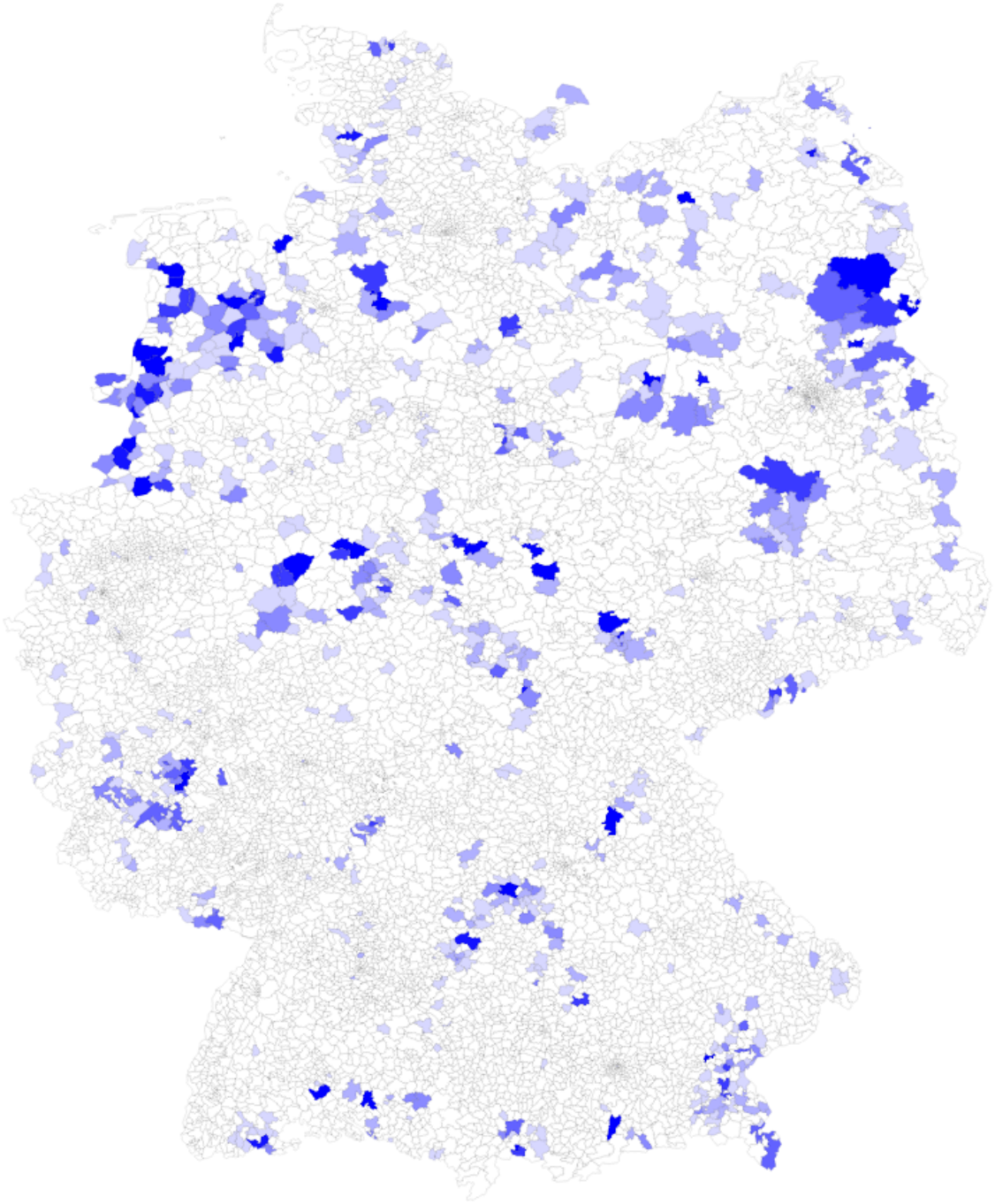

Abbildung 6.2.: Ergebnis der Postleitzahlübersicht. Eigene Abbildung.

wird die Intensitätsübersicht von WASC verwendet. Die Analyse wird mit folgenden Konfigurationsparametern durchgeführt. Der Radius pro Ladesäule wird auf 30km gesetzt. Die Gewichtung jeder Ladesäule ist auf 1 festgelegt.

Die Abbildung 6.3 zeigt die Deutschlandkarte mit einer darübergelegten Heatmap. Die Bereiche bei Köln und Berlin enthalten eine starke rote Färbung. Der größte Teil an grüner Färbung ist um den roten Bereich bei Köln zu finden. In Berlin ist dagegen nur wenig grüne Färbung zu erkennen. Hamburg bis Bremen enthält mehr blaue als grüne Färbung. Im Süden von Deutschland sind vor allem gelbgrüne Ausschläge zu sehen. Aus der Analyse lassen sich einige Folgerungen herauslesen. Die starke Konzentration der Ladesäulen im Westen von Deutschland ist höchstwahrscheinlich durch den Standort der RWE AG zu erklären. Mit dem Sitz in Essen [AG16a] ist auch ein starker Ausbau der Ladesäulen um das Gebiet zu sehen. Die hohe Konzentration in Berlin lässt sich vermutlich einerseits auf Berlin selbst als Großstadt zurückführen, andererseits bietet das Umland aufgrund der kleineren Bevölkerungsdichte weniger Attraktivität für den Ausbau von Ladesäulen. Stuttgart und München sind Sitz für starke Automobilmarken. Dennoch ist dort keine rote Färbung zu sehen, wie bei Berlin oder Köln. Dies liegt vermutlich am stärkeren Ausbau von Ladesäulen anderer Energieunternehmen, wie z.B. EnBW in Stuttgart [EnB16].

6.4. Anbindung an ein Produktivsystem

WASC wurde erfolgreich in ein Managementsystem für Werkstattdaten einer großen deutschen Versicherung integriert und wird produktiv genutzt. Eine Momentaufnahme der Nutzung ist in Abbildung 6.4 zu sehen. WASC wurde dort über die REST-Schnittstelle (Kapitel 5.2.2) angebunden. Dazu wurde im Managementsystem eine neue Seite im bisherigen Stil der Software erstellt. Über Bedienungselemente lassen sich die Konfigurationsparameter von WASC steuern. Diese wurden für den Nutzer vereinfacht. Die Seite kann anschließend eine Anfrage an die REST-Schnittstelle von WASC senden um ein Ergebnis zu erhalten. Im Hintergrund wird die bestehende Datenbank nach den benötigten Datensätzen abgefragt, welche in WASC automatisch in GeoJSON umgewandelt werden. Außerdem wurde auch eine Anbindung über die Frontendschnittstelle (Kapitel 5.2.3) getätigt. Das Frontend ist über den Menüpunkt 'WASC' erreichbar (Abbildung 6.4).

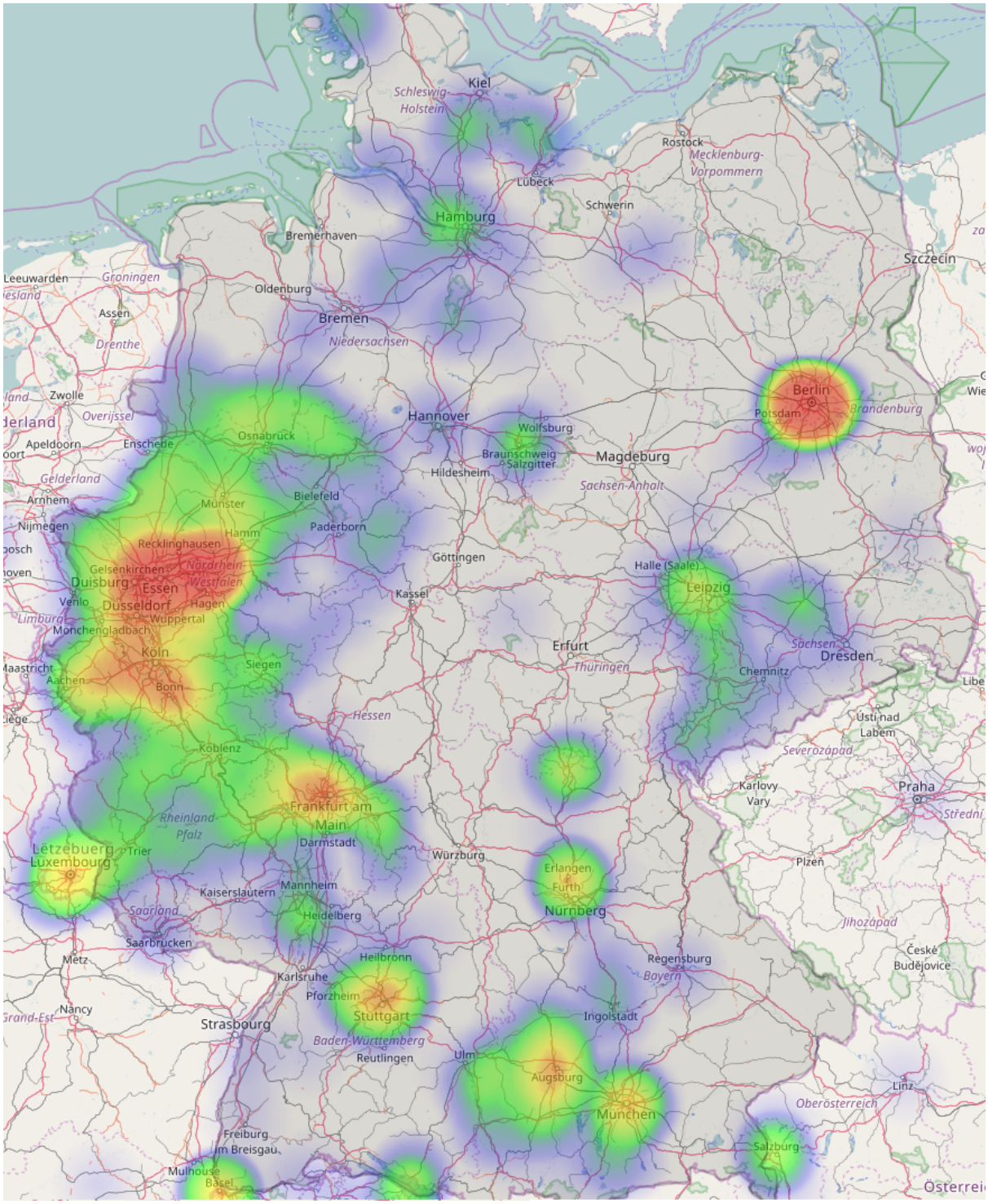

Abbildung 6.3.: Ergebnis der Intensitätsübersicht von Ladesäulen. Eigene Abbildung.

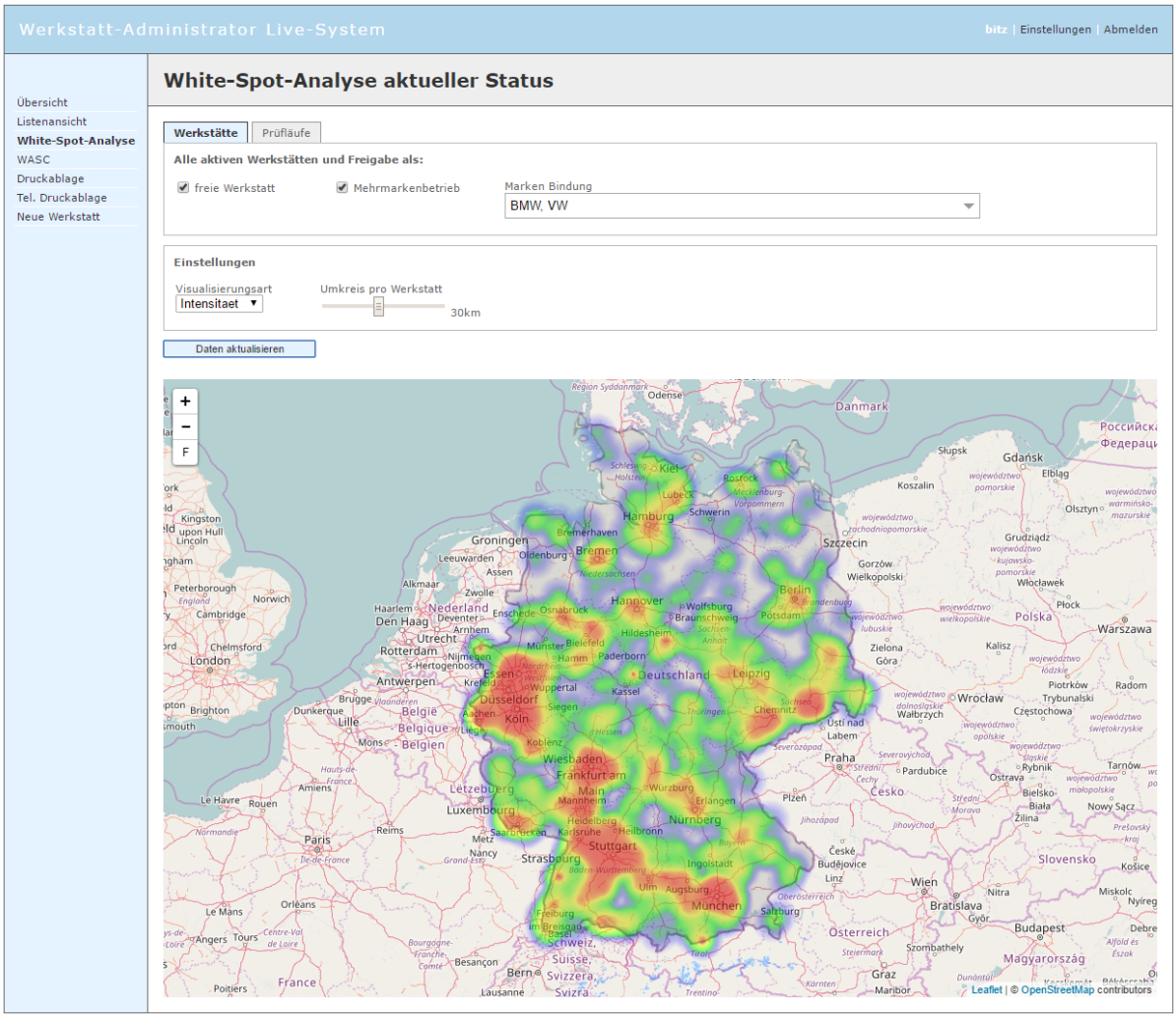

Abbildung 6.4.: Screenshot aus dem Managementsystem für Werkstattdaten mit WASC Anbindung. Eigene Abbildung.

6.5. Verifikation der Anforderungen

In diesem Kapitel wird geprüft, ob WASC alle Anforderungen aus Kapitel 2 erfüllt hat. Dazu wird für jede Anforderung belegt, inwieweit die Anforderung erfüllt ist.

Anwendungsszenario :

WASC wurde erfolgreich in ein Produktivsystem eingebunden (Kapitel 6.4). Das Produktivsystem besitzt Werkstattdaten auf dessen Basis WASC Analysen durchführen kann. Die Anforderung ist erfüllt.

Geodaten einlesen :
: Es werden drei verschiedene Möglichkeiten zur Dateneingabe von WASC angeboten (Kapitel 5.3). Darunter befindet sich ein Standardformat (GeoJSON), ein Datenmodell, sowie eine Möglichkeit zur Datenbankabfragen. Die Anforderung ist erfüllt.

Schnittstellen :
: WASC bietet drei Schnittstellen zur Anbindung an andere Softwareprojekte (Kapitel 5.2). Darunter ist sowohl eine Programmschnittstelle, als auch eine Nutzerschnittstelle in Form einer Weboberfläche enthalten. Die Anforderung ist erfüllt.

Visualisierung der Ergebnisse :
: Alle Ausgabemöglichkeit von WASC basieren auf visueller Darstellung mit geografischen Karten oder einer SVG. Dies ist insbesondere in Kapitel 5.4 verdeutlicht. Die Anforderung ist erfüllt.

Analyse-Algorithmen :
: Im Kapitel 6 wurden drei White-Spot-Analysen durchgeführt. Speziell in Kapitel 6.1 lassen sich 'Lücken' in einem Datensatz finden. Die Anforderung ist erfüllt.

Ergebnisauswertung :
: Hilfsfunktionen sind insbesondere in Abbildung 6.1 und Abbildung 6.2 zu sehen. Eine Zoom Funktion liegt durch die Nutzung von OpenStreetMaps nativ bei. Die Anforderung ist erfüllt.

Evaluation der Software :
: In Kapitel 6.2 und Kapitel 6.1 wurden entsprechende Evaluationen durchgeführt. Die Anforderung ist erfüllt.

Alle Anforderungen an WASC wurden erfüllt und die Verifikation gilt als bestanden. Weiterhin haben sich bei der Arbeit mit WASC folgende Beobachtungen ergeben. Im Produktiveinsatz der Software (Kapitel 6.4) hat sich WASC bei der Arbeit mit statischen geografischen Datenpunkten als flexibel erwiesen. Dies sieht man insbesondere daran, dass nicht nur geografische Datenbestände sich analysieren lassen, sondern auch neu generierte Datensätze nutzbar sind (Kapitel 6.2). Bei der Integration erwies sich die automatische Umwandlung von SQL Ergebnissen in GeoJSON (Kapitel 5.3.2) als zeitsparend. Damit ließen sich Datensätze aus der bestehenden Datenbank mit geringem Aufwand nutzen.

7. Zusammenfassung und Ausblick

Ziel dieser Arbeit war die Entwicklung einer konfigurierbaren Softwarekomponente für White-Spot-Analysen. Nach der Darlegung der Aufgabenstellung und den daraus resultierenden technischen Anforderungen befasste sich die Ausarbeitung zunächst mit den Grundlagen der White-Spot-Analyse, als auch mit der verbundenen Thematik der Geodaten. Dieser Teil der Ausarbeitung bildete die Grundlage für die Entwicklung von WASC.
WASC ist eine durch Parameter einstellbare Softwarekomponente, welche in Webapplikationen integriert werden kann. WASC wurde mit State of the Art Technologien wie einer REST-Schnittstelle und dazugehöriger Swagger Spezifikation, sowie aktuellen Standards wie GeoJSON entwickelt. Unter Zuhilfenahme von OpenStreetMaps und einer Deutschland SVG-Datei entstanden drei Visualisierungsformen, mit welchen sich Geodaten auf Abdeckung, Ballung oder Verteilung pro Postleitzahlgebiet in Deutschland untersuchen lassen. WASC wurde mittels der verfügbaren REST-Schnittstelle erfolgreich in ein Managementsystem für Werkstattdaten einer großen deutschen Versicherung integriert und wird auch produktiv genutzt. Es wurde auch eine Evaluation der Softwarekomponente anhand realer Datensätze vom Fraunhofer IAO und RWE durchgeführt. Dabei hat WASC alle Anforderungen erfüllt.

Ausblick

Aufgrund des Wachstums und der Verbreitung von Smartphones und IoT Geräten wächst nicht nur die Anzahl der verbundenen Geräte, sondern auch die Menge der Daten. Durch das Ausstatten von Geräten mit GPS-Sensoren sind immer mehr Geräte lokalisierbar. Geodaten werden Bestandteil vieler Softwaresysteme, sind aber auch in der Unterhaltungsbranche zu finden. Apps wie Ingress oder Pokemon Go nutzen Geodaten, um den Nutzer in einem virtuellen Abbild seiner Umgebung spielen zu lassen. Die Analyse von Geodaten ist somit ein wichtiger Bestandteil der Informatik. Dabei stellt die White-Spot-Analyse ein nützliches Werkzeug zur Verarbeitung von statischen Geodaten dar. Allerdings entstehen durch neue Themengebiete wie Connected Car viele dynamische Geodaten. Diese können jetzt schon z.B. für eine frühzeitige Erkennung von Staus genutzt werden. Ein Bedarf an neuen Analysemöglichkeiten ist abzusehen. Dafür können bisherige Analysemöglichkeiten wie die White-Spot-Analyse erweitert oder komplett neue Methoden erforscht werden.
Jedoch lässt sich abschließend auch feststellen, dass aufgrund der Existenz von statischen Objekten, wie z.B. Gebäudestandorten, statische Geodaten in absehbarer Zeit ihre Relevanz nicht verlieren werden.

A. Anhang

A.1. Installationshinweis

Im beigefügten Datenträger findet sich neben dieser Arbeit auch das exportierte Eclipse-Projekt der Softwarekomponente WASC. Auf dem Datenträger liegt auch eine README.txt bei, welche genauer beschreibt, wie das Projekt lokal in Betrieb genommen werden kann. Das Exportierte Projekt ist ein Maven-Projekt, welchem mit Java 8 entwickelt wurde.

A.2. Swagger Spezifikation

WASC enthält für seine REST-Schnittstelle eine Swagger Spezifikation. Die Swagger Spezifikation ist im 'WhitespotCore' Projekt unter dem folgenden Pfad zu finden: 'WhitespotCore/src/main/resources/files/swagger/swagger.json'.
Die Spezifikation ist an die Swagger-UI angepasst, dementsprechend finden sich einige Beschreibungen und Informationstexte für den Nutzer der Swagger-UI.

```
{
 "swagger": "2.0",
 "info": {
  "description": "The WASC REST-API offers the same three views as the WASC-Frontend. With the help of swagger
       and swagger-ui it is possible to try the Api right here. **!IMPORTANT NOTE!** After making one Api call,
       please refresh this site, before making another Api call. The Response includes some JS code, what can
       interfere with the new Response.",
  "version": "1.0.0",
  "title": "WASC - REST API"
 },
 "basePath": "/",
 "tags": [
  {
   "name": "CircleMap",
   "description": "cover analysis using OSM"
  },
  {
   "name": "HeatMap",
   "description": "intensity analysis using OSM"
  },
  {
   "name": "PlzMap",
   "description": "postal code analysis with a svg of germany"
  }
 ],
 "schemes": [
```

```
  "https",
     "http"
 ],
 "paths": {
  "/?circlemapraw": {
   "post": {
    "tags": [
     "CircleMap"
    ],
    "summary": "Raw geoJson as input",
    "description": "This Request uses raw GeoJson data as input to get a cover analysis with Open Street Map.",
    "operationId": "circlemapraw",
    "consumes": [
     "application/x-www-form-urlencoded"
    ],
    "produces": [
     "text/html"
    ],
    "parameters": [
     {
      "name": "wasc_api",
      "in": "formData",
      "type": "string",
      "description": "visualisation type",
      "enum": [
       "circlemap"
      ],
      "required": true
     },
     {
      "name": "data",
      "in": "formData",
      "type": "string",
      "default": "",
      "description": "Raw GeoJson Data",
      "required": true
     },
     {
      "name": "radius",
      "in": "formData",
      "description": "Cover radius per point in meters (30000 = 30km) or property key of point.",
      "default": "\"30000\"",
      "type": "string",
      "required": true
     },
     {
      "name": "opacity",
      "description": "Opacity level of each circle.[0-100] (20 means 20%)",
      "in": "formData",
      "default": "\"20\"",
      "type": "string",
      "required": false
     },
     {
      "name": "circlecolor",
      "description": "Hex color for each circle draw",
      "in": "formData",
      "default": "\"#FF0000\"",
      "type": "string",
      "required": false
     },
     {
      "name": "countryshape",
      "type": "string",
      "in": "formData",
```

```
89          "default": "Germany",
90          "description": "Name of the country to highlight. (Germany)",
91          "required": false
92        },
93        {
94          "name": "mapheight",
95          "in": "formData",
96          "type": "string",
97          "default": "\"750\"",
98          "description": "The height of the map to display in px (css). (750 means 750px)",
99          "required": false
100       }
101     ],
102     "responses": {
103       "405": {
104         "description": "Invalid input",
105         "schema": {
106           "$ref": "#/definitions/errormsg"
107         }
108       }
109     }
110   }
111 },
112 "/?circlemapdb": {
113   "post": {
114     "tags": [
115       "CircleMap"
116     ],
117     "summary": "SQL query as Input",
118     "description": "This Request uses a SQL query to get data. By defining a longitude and latitude field for
          the SQL table, it is possible to generate GeoJson data on the serverside. This data is used to create
          a cover analysis with Open Street Maps.",
119     "operationId": "circlemapdb",
120     "consumes": [
121       "application/x-www-form-urlencoded"
122     ],
123     "produces": [
124       "text/html"
125     ],
126     "parameters": [
127       {
128         "name": "wasc_api",
129         "in": "formData",
130         "type": "string",
131         "description": "visualisation type",
132         "enum": [
133           "circlemap"
134         ],
135         "required": true
136       },
137       {
138         "name": "query",
139         "in": "formData",
140         "type": "string",
141         "default": "",
142         "description": "SQL query (WASC needs to have a connection configured!)",
143         "required": true
144       },
145       {
146         "name": "lat_field",
147         "in": "formData",
148         "description": "latitude field key of the SQL table",
149         "default": "",
150         "type": "string",
151         "required": true
```

```
      },
      {
        "name": "lng_field",
        "in": "formData",
        "description": "longitude field key of the SQL table",
        "default": "",
        "type": "string",
        "required": true
      },
      {
        "name": "radius",
        "in": "formData",
        "description": "Cover radius per point in meters (30000 = 30km) or property key of point.",
        "default": "\"30000\"",
        "type": "string",
        "required": true
      },
      {
        "name": "opacity",
        "description": "Opacity level of each circle.[0-100] (20 means 20%)",
        "in": "formData",
        "default": "\"20\"",
        "type": "string",
        "required": false
      },
      {
        "name": "circlecolor",
        "description": "Hex color for each circle draw",
        "in": "formData",
        "default": "\"#FF0000\"",
        "type": "string",
        "required": false
      },
      {
        "name": "countryshape",
        "type": "string",
        "in": "formData",
        "default": "Germany",
        "description": "Name of the country to highlight. (Germany)",
        "required": false
      },
      {
        "name": "mapheight",
        "in": "formData",
        "type": "string",
        "default": "\"750\"",
        "description": "The height of the map to display in px (css). (750 means 750px)",
        "required": false
      }
    ],
    "responses": {
      "405": {
        "description": "Invalid input",
        "schema": {
          "$ref": "#/definitions/errormsg"
        }
      }
    }
  }
},
"/?heatmapraw": {
  "post": {
    "tags": [
      "HeatMap"
    ],
```

```
      "summary": "Raw geoJson as input",
      "description": "This Request uses raw GeoJson data as input to get a intensity analysis with Open Street
          Map.",
      "operationId": "heatmapraw",
      "consumes": [
        "application/x-www-form-urlencoded"
      ],
      "produces": [
        "text/html"
      ],
      "parameters": [
        {
          "name": "wasc_api",
          "in": "formData",
          "type": "string",
          "description": "visualisation type",
          "enum": [
            "heatmap"
          ],
          "required": true
        },
        {
          "name": "data",
          "in": "formData",
          "type": "string",
          "default": "",
                   "description": "Raw GeoJson Data",
          "required": true
        },
        {
          "name": "radius",
          "in": "formData",
          "description": "Cover radius per point in meters (30000 = 30km) or property key of point.",
          "default": "\"30000\"",
          "type": "string",
          "required": true
        },
        {
          "name": "weight",
          "description": "property key of point (if point has specifix weight/priority)",
          "in": "formData",
          "default": "",
          "type": "string",
          "required": false
        },
        {
          "name": "countryshape",
          "type": "string",
          "in": "formData",
          "default": "Germany",
          "description": "Name of the country to highlight. (Germany)",
          "required": false
        },
        {
          "name": "mapheight",
          "in": "formData",
          "type": "string",
          "default": "\"750\"",
          "description": "The height of the map to display in px (css). (750 means 750px)",
          "required": false
        }
      ],
      "responses": {
        "405": {
          "description": "Invalid input",
```

```
        "schema": {
          "$ref": "#/definitions/errormsg"
        }
       }
      }
     }
    },
    "/?heatmapdb": {
     "post": {
      "tags": [
       "HeatMap"
      ],
      "summary": "SQL query as Input",
      "description": "This Request uses a SQL query to get data. By defining a longitude and latitude field for
           the SQL table, it is possible to generate GeoJson data on the serverside. This data is used to create
           a intensity analysis with Open Street Maps.",
      "operationId": "heatmapdb",
      "consumes": [
       "application/x-www-form-urlencoded"
      ],
      "produces": [
       "text/html"
      ],
      "parameters": [
       {
        "name": "wasc_api",
        "in": "formData",
        "type": "string",
        "description": "visualisation type",
        "enum": [
         "heatmap"
        ],
        "required": true
       },
       {
        "name": "query",
        "in": "formData",
        "type": "string",
        "default": "",
        "description": "SQL query (WASC needs to have a connection configured!)",
        "required": true
       },
       {
        "name": "lat_field",
        "in": "formData",
        "description": "latitude field key of the SQL table",
        "default": "",
        "type": "string",
        "required": true
       },
       {
        "name": "lng_field",
        "in": "formData",
        "description": "longitude field key of the SQL table",
        "default": "",
        "type": "string",
        "required": true
       },
       {
        "name": "radius",
        "in": "formData",
        "description": "Cover radius per point in meters (30000 = 30km) or property key of point.",
        "default": "\"30000\"",
        "type": "string",
        "required": true
```

```
344         },
345         {
346           "name": "weight",
347           "description": "property key of point (if point has specifix weight/priority)",
348           "in": "formData",
349           "default": "",
350           "type": "string",
351           "required": false
352         },
353         {
354           "name": "countryshape",
355           "type": "string",
356           "in": "formData",
357           "default": "Germany",
358           "description": "Name of the country to highlight. (Germany)",
359           "required": false
360         },
361         {
362           "name": "mapheight",
363           "in": "formData",
364           "type": "string",
365           "default": "\"750\"",
366           "description": "The height of the map to display in px (css). (750 means 750px)",
367           "required": false
368         }
369       ],
370       "responses": {
371         "405": {
372           "description": "Invalid input",
373           "schema": {
374             "$ref": "#/definitions/errormsg"
375           }
376         }
377       }
378     }
379   },
380   "/?plzmapraw": {
381     "post": {
382       "tags": [
383         "PlzMap"
384       ],
385       "summary": "Raw geoJson as input",
386       "description": "**!IMPORTANT NOTE!** Response Data is about 17Mb big. Be patien with receiving! This
          Request uses raw GeoJson data as input to get a postal code analysis in germany with a svg.",
387       "operationId": "plzmapraw",
388       "consumes": [
389         "application/x-www-form-urlencoded"
390       ],
391       "produces": [
392         "text/html"
393       ],
394       "parameters": [
395         {
396           "name": "wasc_api",
397           "in": "formData",
398           "type": "string",
399           "description": "visualisation type",
400           "enum": [
401             "plzmap"
402           ],
403           "required": true
404         },
405         {
406           "name": "data",
407           "in": "formData",
```

```
        "type": "string",
        "default": "",
                "description": "Raw GeoJson Data",
        "required": true
      },
      {
        "name": "plzkey",
        "in": "formData",
        "description": "property key of the postal code of each point",
        "default": "",
        "type": "string",
        "required": true
      },
      {
        "name": "colormin",
        "description": "hex color for postal code areas with low element coverage",
        "in": "formData",
        "default": "#b3c6ff",
        "type": "string",
        "required": false
      },
      {
        "name": "colormax",
        "type": "string",
        "in": "formData",
        "default": "#002db3",
        "description": "hex color for postal code areas with high element coverage",
        "required": false
      },
      {
        "name": "colormid",
        "in": "formData",
        "type": "string",
        "default": "",
        "description": "hex color for postal code areas between low and high coverage",
        "required": false
      }
    ],
    "responses": {
      "400": {
        "description": "Invalid input",
        "schema": {
          "$ref": "#/definitions/errormsg"
        }
      }
    }
  }
 },
 "/?plzmapdb": {
  "post": {
   "tags": [
    "PlzMap"
   ],
   "summary": "SQL query as Input",
   "description": "**!IMPORTANT NOTE!** Response Data is about 17Mb big. Be patien with receiving! This
       Request uses a SQL query to get data. By defining a longitude and latitude field for the SQL table,
       it is possible to generate GeoJson data on the serverside. This data is used to create a postal code
       analysis in germany with a svg.\n !IMPORTANT NOTE! Response Data is about 17Mb big. Be patien with
       receiving!",
   "operationId": "plzmapdb",
   "consumes": [
    "application/x-www-form-urlencoded"
   ],
   "produces": [
    "text/html"
```

```
      ],
      "parameters": [
        {
          "name": "wasc_api",
          "in": "formData",
          "type": "string",
          "description": "visualisation type",
          "enum": [
            "plzmap"
          ],
          "required": true
        },
        {
          "name": "query",
          "in": "formData",
          "type": "string",
          "default": "",
          "description": "SQL query (WASC needs to have a connection configured!)",
          "required": true
        },
        {
          "name": "lat_field",
          "in": "formData",
          "description": "latitude field key of the SQL table",
          "default": "",
          "type": "string",
          "required": true
        },
        {
          "name": "lng_field",
          "in": "formData",
          "description": "longitude field key of the SQL table",
          "default": "",
          "type": "string",
          "required": true
        },
        {
          "name": "plzkey",
          "in": "formData",
          "description": "property key of the postal code of each point",
          "default": "",
          "type": "string",
          "required": true
        },
        {
          "name": "colormin",
          "description": "hex color for postal code areas with low element coverage",
          "in": "formData",
          "default": "#b3c6ff",
          "type": "string",
          "required": false
        },
        {
          "name": "colormax",
          "type": "string",
          "in": "formData",
          "default": "#002db3",
          "description": "hex color for postal code areas with high element coverage",
          "required": false
        },
        {
          "name": "colormid",
          "in": "formData",
          "type": "string",
          "default": "",
```

```
534         "description": "hex color for postal code areas between low and high coverage",
535         "required": false
536       }
537      ],
538      "responses": {
539       "405": {
540         "description": "Invalid input",
541         "schema": {
542           "$ref": "#/definitions/errormsg"
543         }
544       }
545      }
546     }
547    }
548  },
549  "definitions": {
550    "errormsg": {
551      "type": "object",
552     "properties": {
553       "error": {
554         "description": "Error message",
555         "type": "string"
556       }
557     }
558    }
559  }
560 }
```

Literaturverzeichnis

[AB16] F. AB. *Flightradar24*. 2016. URL: https://www.flightradar24.com/ (zitiert auf S. 22, 23).

[AG16a] R. AG. *RWE Impressum*. 2016. URL: http://www.rwe.com/web/cms/de/1494828/rwe/impressum/ (zitiert auf S. 47).

[AG16b] R. AG. *RWE Ladesaeulendatensatz im, JSON Format*. 2016. URL: https://www.rwe-mobility.com/emobilityfrontend/rs/chargingstations/ (zitiert auf S. 43, 45).

[AG16c] R. AG. *RWE Ladesaeulenfinder*. 2016. URL: https://www.rwe-mobility.com/web/cms/de/1195202/emobility/rwe-ladesaeulenfinder/ (zitiert auf S. 43, 45).

[Aga15a] V. Agafonkin. *Leaflet*. 2015. URL: http://leafletjs.com/index.html (zitiert auf S. 40).

[Aga15b] V. Agafonkin. *Leaflet GeoJSON Example*. 2015. URL: http://leafletjs.com/examples/geojson.html (zitiert auf S. 40).

[Aga15c] V. Agafonkin. *Leaflet Plugin System*. 2015. URL: http://leafletjs.com/plugins.html (zitiert auf S. 40).

[Ava16] Avanzu.de. *Umkreissuche per SQL Query*. 2016. URL: http://www.avanzu.de/php-entwicklung/howtos/umkreissuche-mit-sql-und-php/ (zitiert auf S. 14).

[Ber13] M. Berger. *Vomper Loch (47.35N,11.52E) als Relief, Höhendaten von ASTER*. 2013. URL: https://commons.wikimedia.org/wiki/File:Schachbrett_Nummerierung.png (zitiert auf S. 21).

[Ber15] gb consite GmbH Geomarketing Software und Beratung. *Zukünftige Unternehmensstandorte mit der White Spot Analyse*. 2015. URL: http://www.gbconsite.de/standortsuche-white-spot-analyse.php (zitiert auf S. 13).

[EnB16] EnBW. *EnBW-Ladestationen*. 2016. URL: https://www.enbw.com/privatkunden/energie-und-zukunft/e-mobilitaet/ladestationen/ (zitiert auf S. 47).

[geo16] geotools.org. *GeoTools Java code library*. 2016. URL: http://geotools.org/about.html (zitiert auf S. 42).

[Gie06] I. Giel. *Schachbrett Nummerierung.png*. 2006. URL: https://commons.wikimedia.org/wiki/File:Schachbrett_Nummerierung.png (zitiert auf S. 20).

[Goo16a] Google. *Google Earth*. 2016. URL: https://www.google.com/intl/de/earth/ (zitiert auf S. 22).

[Goo16b] Google. *Keyhole Markup Language*. 2016. URL: https://developers.google.com/kml/ (zitiert auf S. 25).

[Goo16c] Google. *Keyhole Markup Language Reference*. 2016. URL: https://developers.google.com/kml/documentation/kmlreference (zitiert auf S. 25, 27).

[HB+08a] A. D. Howard Butler Martin Daly et al. *The GeoJSON Format*. 2008. URL: http://geojson.org/ (zitiert auf S. 26).

[HB+08b] A. D. Howard Butler Martin Daly et al. *The GeoJSON Format Specification*. 2008. URL: http://geojson.org/geojson-spec.html (zitiert auf S. 26).

[inn16] innogy. *eMobility*. 2016. URL: https://www.innogy.com/web/cms/de/3020064/home/emobility/mobilitaet/ (zitiert auf S. 43).

[jso16] json.org. *ECMA-404 The JSON Data Interchange Standard.* 2016. URL: http://www.json.org/ (zitiert auf S. 24).

[Neu15] T. Neuhetzki. *Telekom-LTE-Netz: 95 Prozent Netzabdeckung bis 2018*. 2015. URL: https://www.teltarif.de/telekom-lte-vdsl-netzausbau-hoettges-netzabdeckung/news/58800.html (zitiert auf S. 7).

[NPE15] Nationale-Plattform-Elektromobilitaet. *Ladeinfrastruktur für Elektrofahrzeuge in Deutschland - Statusbericht und Handlungsempfehlungen 2015*. 2015. URL: http://nationale-plattform-elektromobilitaet.de/fileadmin/user_upload/Redaktion/NPE_AG3_Statusbericht_LIS_2015_barr_bf.pdf (zitiert auf S. 7).

[Swa16a] Swagger.io. *Swagger UI*. 2016. URL: http://swagger.io/swagger-ui/ (zitiert auf S. 41).

[Swa16b] Swagger.io. *Swagger Webseite*. 2016. URL: http://swagger.io/ (zitiert auf S. 40).

[Vod16] Vodafone. *Netzabdeckung: Detail-Ansicht*. 2016. URL: https://www.vodafone.de/privat/hilfe-support/detail-ansicht-netzabdeckung.html (zitiert auf S. 10).

[Wü16] P. D. K. Wübbenhorst. *Bedarfserkennung*. 2016. URL: http://wirtschaftslexikon.gabler.de/Archiv/6011/bedarfserkennung-v7.html (zitiert auf S. 13).

[Ber64] Berndt, Georg and Trumpold, Harry. „Winkeleinheiten“. In: *Technische Winkelmessungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1964, S. 6–8 (zitiert auf S. 14).

[DAS07] DAS EUROPÄISCHE PARLAMENT. *RICHTLINIE 2007/2/EG DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 14. März 2007 zur Schaffung einer Geodateninfrastruktur in der Europäischen Gemeinschaft (INSPIRE)*. 2007, 14 Seiten (zitiert auf S. 19).

[Daz12] Dazer Michael. *RESTful APIs - Eine Übersicht*. Technical University Berlin Germany, 2012, 8 Seiten (zitiert auf S. 31).

[Dr.08] Dr. Basudeb Bhatta. *Remote sensing and GIS*. 1. publ. New Delhi [u.a.]: Oxford University Press (India), 2008, XVIII, 685 Seiten , 16 Taf. (Zitiert auf S. 14).

[Fra09] Fraunhofer IAO. *IT-gestützte White-Spot-Analyse*. 2009. URL: http://wiki.iao.fraunhofer.de/images/studien/studie-it-gestuetzte-white-spot-analyse_fraunhofer-iao.pdf (zitiert auf S. 13).

[Gol07] Goldberg, Daniel W. and Wilson, John P. and Knoblock, Craig A. „A Geocoding Best Practices Guide". In: *Journal of the Urban and Regional Information Systems Association*. 2007, 287 pages (zitiert auf S. 16).

[Ins03] Institut für Kartographie und Geoinformation Universität Bonn. *Geoinformation und ihre Dimensionen*. Universität Bonn und geoinformation.net, 2003, 13 Seiten (zitiert auf S. 19–21).

[Int13] International Organization for Standardization. *Geographic information — Data quality (ISO 19157:2013(en))*. 2013 (zitiert auf S. 23).

[Joo00] Joos, Gerhard. „Zur Qualität von objektstrukturierten Geodaten". In: *Schriftenreihe des Instituts Heft 66*. Universität der Bundeswehr München, Fakultät für Bauingenieur- und Vermessungswesen, 2000 (zitiert auf S. 23, 24).

[Mar16] Martin Kompf. *Entfernungsberechnung zwischen zwei Geokoordinaten*. 2016. URL: https://www.kompf.de/gps/distcalc.html (zitiert auf S. 14).

[Moz15] Mozilla Developer Network. *Scalable Vector Graphics (SVG)*. 2015. URL: https://developer.mozilla.org/en-US/docs/Web/SVG (zitiert auf S. 11).

[Pro01] Professor Dr. Wilhelm Raith Universität Bielefeld and others. *Lehrbuch der Experimentalphysik, Bd.7, Erde und Planeten*. Begundene Ausgabe Band 7, 2.Auflage. De Gruyter, 2001, 742 Seiten (zitiert auf S. 14).

[Pro05] Professor Dr. Norbert Bartelme. *Geoinformatik: Modelle • Strukturen • Funktionen*. 4., vollständig überarbeitete Auflage. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005 (zitiert auf S. 13, 23, 24).

[Pro10a] Prof. Dr.-Ing. Ralf Bill. *Grundlagen der Geo-Informationssysteme*. 5. Auflage. Universität Rostock, Wichmann Verlag, 2010, 809 Seiten (zitiert auf S. 19).

[Pro10b] Prof. Dr.-Ing. Wolfgang Reinhardt. *Vorlesung: Bedeutung von Metadaten und Qualität von Daten*. 2010. URL: https://www.unibw.de/inf4/professuren/geoinformatik/lehre/skripten/skripte/skripten_wt_10/vorl_gis_1_kap_5.pdf (zitiert auf S. 23, 24).

[Sch08] Schertenleib, M.H. and Egli-Brož, H. *Grundlagen Geografie: Aufgaben des Fachs, Erde als Himmelskörper und Kartografie*. Compendio Bildungsmedien, 2008, 216 Seiten (zitiert auf S. 14).

[Sch10] Schulte, Bennet and Lippmann, Frank. „Geokodierung mit Webkartendiensten – Möglichkeiten , Unterschiede und Grenzen Einführung Georeferenzierung von Anschriften Adresskodierung im Internet". In: *Angewandte Geoinformatik 2010* (2010), S. 773–778 (zitiert auf S. 16).

[Tim06] Tim Bray, Jean Paoli, Eve Maler and others. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. 2006. URL: https://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals (zitiert auf S. 24).

[Tua12] Tuan, Tuan Anh and Vinh, Phuoc Tran and Duy, Huynh K. „A study on 4D GIS spatio-temporal data model“. In: *Proceedings - 4th International Conference on Knowledge and Systems Engineering, KSE 2012* (2012) (zitiert auf S. 22).

Alle URLs wurden zuletzt am 5. 12. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift