Konzeption und prototypische Realisierung einer IT-Infrastruktur zur automatisierten Bereitstellung und Darstellung von Umweltmessdaten in Webanwendungen auf Basis aktueller Internettechnologien

Bachelorarbeit

für die Prüfung zum

Bachelor of Engineering

des Studiengangs Informationstechnik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Claus Burkhart

August 2016

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	8864278, TINF13B3
Ausbildungsfirma	LUBW Landesanstalt für Umwelt, Messung und Naturschutz Baden-Württemberg, Karlsruhe
Betreuer	M. Sc. Wirtschaftsinformatik Daniel Kimmig
Gutachter	Dipl. Inform. Thorsten Schlachter

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: *Konzeption und prototypische Realisierung einer IT-Infrastruktur zur automatisierten Bereitstellung und Darstellung von Umweltmessdaten in Webanwendungen auf Basis aktueller Internettechnologien* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, August 2016

Claus Burkhart

Abstract

The aim of this thesis is to design a software architecture for automated provisioning of measurement data and their visual representation in web applications. The results of this thesis will contribute the ongoing activity of migrating the LUBW website onto a new portal server technology. During that process it becomes apparent that pages, where measurement data are represented, have to be considered separately, because they have special requirements. As the first step an analysis of two existing websites, offering air and noise measurement data, was conducted. Based on this analysis a new concept describing the new software architecture as well as possible components was formulated. For two example scenarios, prototypes were developed to validate the concept. These prototypes already can provide highly interactive web-based charts of current measurement data. To select supporting technologies, an accompanying literature review of current web technologies was performed.

Zusammenfassung

Ziel dieser Arbeit ist die Konzeption einer Software-Architektur zur automatisierten Bereitstellung von Messdaten und deren Darstellung in Webanwendungen. Die Ergebnisse dieser Thesis haben direkten Einfluss auf die momentan laufende Migration der LUBW-Webseiten auf eine neue Portalserver-Technologie. Während dieser Umstellung stellte sich heraus, dass Seiten, auf denen Messdaten dargestellt werden, gesondert betrachtet werden müssen, da sie spezielle Anforderungen haben. Zunächst wurde dazu eine Analyse von zwei bestehenden Webangeboten durchgeführt. Basierend auf dieser Analyse wurde ein Konzept formuliert, welches sowohl eine neue Software-Architektur als auch mögliche Komponenten beschreibt. Zur Validierung des erstellten Konzeptes wurden für zwei Beispielszenarien Prototypen entwickelt, die bereits ansprechende Diagramme von aktuellen Messdaten liefern. Um geeignete Technologien auswählen zu können, wurde begleitend eine wissenschaftliche Literaturrecherche von aktuellen Internettechnologien durchgeführt.

Inhaltsverzeichnis

Abkürzungsverzeichnis

BASE	Basically Available, Soft State, Eventually Consistent
BITBW	Landesoberbehörde IT Baden-Württemberg
CMS	Content-Management-System
DBMS	Datenbankmanagementsysteme
DWD	Deutscher Wetterdienst
ETL	Extract, Transform, Load
FTP	File Transfer Protocol
GUI	Graphical User Interface
HATEOAS	Hypermedia as the engine of application state
HDFS	Hadoop Distributed File System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ITZ	Informationstechnisches Zentrum
JVM	Java Vitual Machine
KIT	Karlsruher Institut für Technologie
LUBW	Landesanstalt für Umwelt, Messung und Naturschutz Baden-Württemberg
LUPO	Landesumweltportal Baden-Württemberg
MNZ	Messnetz-Zentrale Baden-Württemberg
MVCC	Multiversion Concurrency Control
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TSDB	Time Series Database
UIS	Umweltinformationssystem Baden-Württemberg
UM	Ministerium für Umwelt, Klima und Energiewirtschaft Baden-Württemberg
URI	Uniform Resource Identifier
VML	Vector Markup Language

Abbildungsverzeichnis

Listingverzeichnis

1 Einleitung

Die Landesanstalt für Umwelt, Messung und Naturschutz Baden-Württemberg (LUBW) ist eine dem Ministerium für Umwelt, Klima und Energiewirtschaft Baden-Württemberg (UM) nachgeordnete Behörde. Sie ist Hauptanlaufstelle für Fragen rund um die Themen Umwelt-Naturschutz, Strahlenschutz, Arbeitsschutz und der Produktionssicherheit in Baden-Württemberg. Die LUBW beschäftigt rund 550 Fachkräfte und betreibt umfangreiche Messnetze, die Daten über den Zustand von Luft, Wasser und Boden liefern. Des weiteren betreibt die LUBW das Umweltinformationssystem Baden-Württemberg (UIS), welches als Teil des E-Government-Konzepts des Landes moderne leistungsfähigere Informationstechnik bereitstellt, mit deren Hilfe umfangreiche Umweltdatenbestände aus vielen Quellen erschlossen und wirkungsvoll an verschiedene Zielgruppen vermittelt werden.

Aufgabenbereich ist das Informationstechnisches Zentrum (ITZ), genauer gesagt in Referat 53 „Übergreifende Umweltanwendungen“, Sachgebiet 53.1 „Umweltportale“. Dieses Sachgebiet beschäftigt sich u.a. mit dem Aufbau und dem Betrieb vom Landesumweltportal Baden-Württemberg (LUPO) und der App „Meine Umwelt“, sowie weiterer Portale und Apps. Außerdem ist es zuständig für den Betrieb der Internet- und Intranet-Auftritte der LUBW sowie des UIS-Landesintranets.

1.1 Motivation und Zielsetzung

Aktuell werden Messdaten (z.B. Luftmesswerte, Pegelwerte) der LUBW mit für jede Anwendungssituation spezifischen Lösungen und Webtechniken im Internet dargestellt.

Dabei entsprechen die Umsetzungen z.T. nicht mehr dem Stand der Technik, insbesondere fehlen generischen Mechanismen zur performanten und automatisierten Bereitstellung der Daten. Außerdem unterstützen die Datenvisualisierungen kaum dynamische und interaktive Elemente und sie passen sich nicht adaptiv an verschiedene Gerätetypen an.

Aus diesen Gründen wurde in der Projektarbeit „Analyse der Möglichkeiten für die Modernisierung der Darstellung und Visualisierung von Umweltmessdaten mit neuen

Webtechnologien“ bereits für die Fach-Referate 31 (Luftqualität) und 34 (Technischer Arbeitsschutz) der Fluss der jeweiligen Messdaten bis zur Darstellung im Internet beschrieben und deren Darstellungsformen analysiert.

Aufbauend auf die Ergebnisse dieses Projekts ergibt sich für diese Bachelorarbeit folgende Zielsetzung:

Erarbeitung eines konkreten Konzeptes, wie verschiedene Typen von Messdaten mit Hilfe einer generischen Infrastruktur zur Bereitstellung der Daten, sowie von HTML5-Technologien performant und hochverfügbar im Internet dargestellt werden können. Begleitend dazu sollen geeignete Technologien und Dienste evaluiert werden. Zur Validierung des Konzepts soll für verschiedene Beispielszenarien der Datenfluss und die jeweilige Visualisierungsart prototypisch umgesetzt werden.

1.2 Aufbau der Arbeit

Nach einer Einführung in die Thematik in Kapitel 1 wird in Kapitel 2 ein Überblick über die beiden Webangebote „Luftqualität aktuell“ und „Verkehrsgeräusche aktuell“ gegeben und daraus konkrete Anforderungen abgeleitet.

Kapitel 3 stellt eine Analyse dar, um verschiedene Technologien und Dienste dahingehend zu untersuchen, inwieweit sie bei einer Erneuerung der zuvor beschriebenen Webangebote eingesetzt werden können.

In Kapitel 4 wird darauf aufbauend eine Software-Architektur beschrieben, mit der die Umweltmessdaten der unterschiedlichen Fachbereiche der LUBW im Internet ansprechend und hoch verfügbar präsentieren werden können.

Die Beschreibung der prototypischen Umsetzung dieser Architektur an zwei Beispielszenarien erfolgt in Kapitel 5.

Ein Fazit sowie eine Ausblick auf zukünftige Tätigkeiten in Kapitel 6 runden diese Arbeit ab.

2 Randbedingungen und Anforderungen

Um bei der Konzeption einer neuen IT-Infrastruktur zur Bereitstellung und Darstellung von Umweltmessdaten möglichst geeignete Anknüpfungspunkte an die bestehenden Strukturen identifizieren zu können, werden in diesem Kapitel zunächst die momentanen Gegebenheiten und Randbedingungen beschrieben.

Auf Basis dieser Ist-Analyse werden im zweiten Abschnitt konkrete Anforderungen für verschiedene Nutzergruppen abgeleitet, die bei einer Erneuerung des Webangebots beachtet werden müssen.

2.1 Ist-Zustand

Die beiden ersten Unterabschnitte geben einen Überblick über die Daten und ihre Darstellungsformen in den bestehenden Internetauftritte der Fachbereiche Luft (Luftqualität aktuell[1]) und Lärm (Verkehrsgeräusche aktuell[2]). Der letzte Unterabschnitt beschreibt allgemeine Bedingungen, die bei der Umstellung eines LUBW-Webangebots beachtet werden müssen. Neben den genannten Fachbereichen existieren noch die Fachbereiche Hochwasser und Radioaktivität, die nicht Teil dieser Ausarbeitung sind. Die beschriebenen Anforderungen sind aber auch auf diese Bereiche übertragbar.

2.1.1 Luftqualität aktuell

Zur Überwachung der Luftqualität in Baden-Württemberg betreibt die LUBW drei Messnetze. Zur Messung von Schadstoffen in der Luft gibt es das Luftmessnetz und das Spotmessnetz, wobei letzteres speziell straßennahe und somit hochbelastete Punkte erfasst. Zur Messung von Stoffen, die sich auf dem Erdboden abgelagert haben, gibt es das Depositionsnetz.

Alle erhobenen Daten laufen zusammen in der Messnetz-Zentrale Baden-Württemberg (MNZ). Hier ist man zuständig für die Plausibilisierung und langfristige Speicherung der

[1] http://mnz.lubw.baden-wuerttemberg.de/messwerte/aktuell/
[2] http://www4.lubw.baden-wuerttemberg.de/servlet/is/253556/

Daten. Außerdem werden von hier aus Daten an verschiedene Stellen (z.B. der DWD oder das UIS) weitergeleitet.

Zur Veröffentlichung aktueller Daten des Luftmessnetzes im Internet werden momentan die an den Stationen gemessenen Schadstoffe (im Folgenden auch als Komponenten bezeichnet) stündlich per FTP in einer CSV-Datei auf einen Rechner in der MNZ gepusht. Diese Datei enthält zu allen Komponenten die jeweiligen 30-Minuten-Werte des aktuellen Tages und der vorherigen elf Tage.

Momentan werden für jede Station, für alle gemessenen Komponenten, Werte des aktuellen und der vorherigen Tages in einer Tabelle angezeigt (vgl. Abb. 2.1). Des weiteren gibt es unterschiedliche Übersichtstabellen, wie beispielsweise alle Stationen, die in einem Regierungsbezirk liegen oder die eine bestimmte Komponente messen.

Karlsruhe-Nordwest (vorläufige Daten) **15.08.2016 09:00**

		15.08.2016		14.08.2016				
Komponente	**Mittelungs-zeitraum**	**Aktueller Messwert**	**Tages-Maximum**	**Tages-Maximum**	**Tages-Mittelwert**	**Immissions-Werte**	**Anzahl Überschr.**	**Einheit**
Feinstaub (PM10)	24-Std.	17	17	17	17	50 1)	1	µg/m³
Stickstoffdioxid (NO_2)	1-Std.	21	32	29	18	200 2)	0	µg/m³
Ozon (O_3)	1-Std.	32	51	123	69	180 3)		µg/m³
	8-Std.	31	91	117		120 4)		µg/m³
Schwefeldioxid (SO_2)	1-Std.	2	2	17	4	350 5)	0	µg/m³

Abbildung 2.1: Tabelle im aktuellen Internetauftritt „Luftqualität aktuell“

In ersten Besprechungen das Ziel definiert die bestehenden Strukturen stark zu vereinfachen, d.h. die Anzahl von Tabelle auf wenige mit möglichst gleichem Aufbau zu verringern. Angedacht ist beispielsweise eine Tabelle für jede Komponente, in der nur der aktuelle Stundenwert und der Tageshöchstwert für jeweils alle Stationen angezeigt werden, die diesen Schadstoff messen.

Um Verläufe der unterschiedlichen Komponenten zu veranschaulichen, werden Säulendiagramme eingesetzt, wobei die Werte des aktuellen Tages und die der acht vorhergehenden Tage dargestellt werden. Je nach Höhe der Werte, werden die Säulen dabei unterschiedlich eingefärbt (vgl. Abb. 2.2).

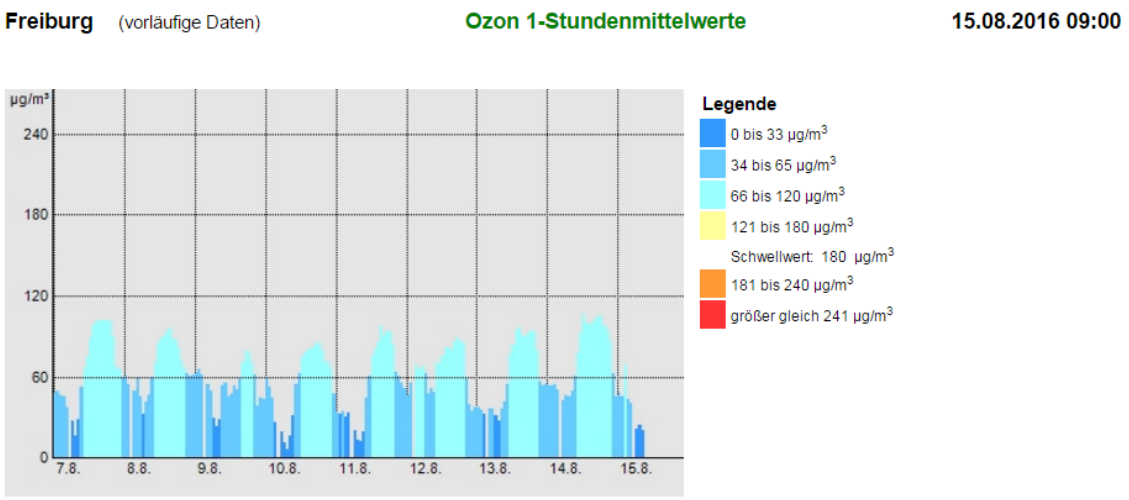

Abbildung 2.2: Verlaufsgrafik im aktuellen Internetauftritt „Luftqualität aktuell“

Karten werden auf den Luft-Seiten eingesetzt, um sich einen Überblick über die Lage der einzelnen Messstationen verschaffen zu können und um die Verteilung von Immissionen flächenhaft zu veranschaulichen (vgl. Abb. 2.3).

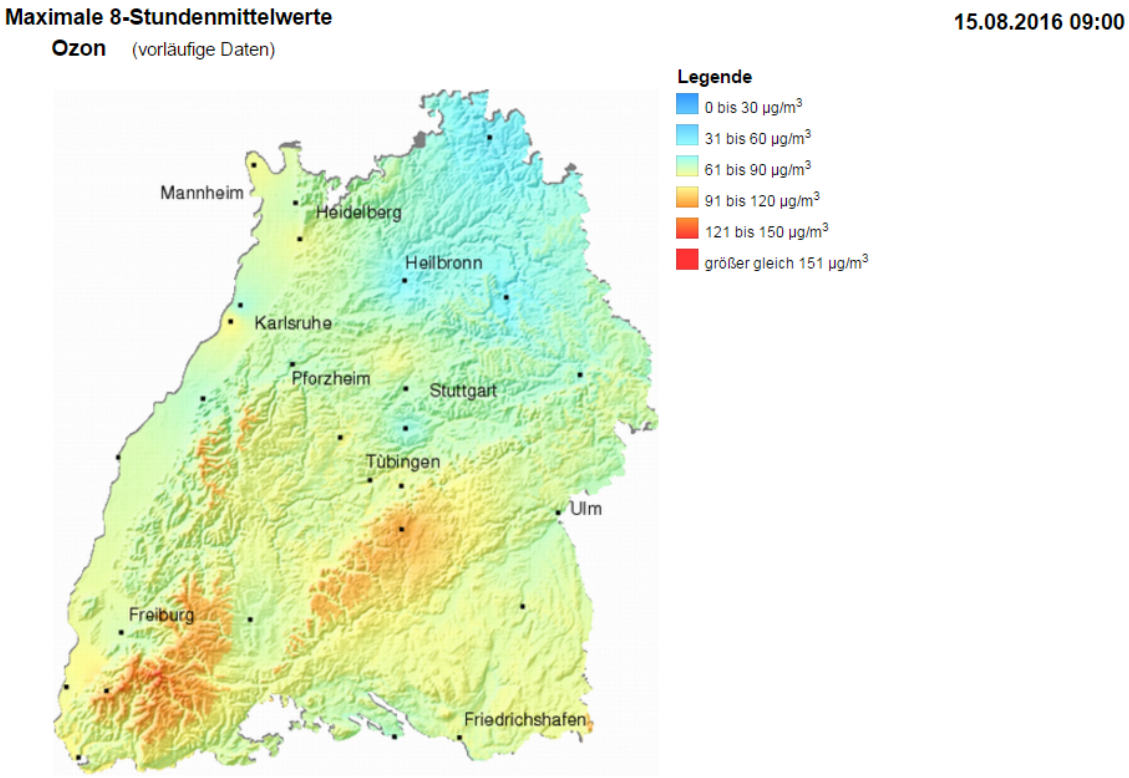

Abbildung 2.3: Kartendarstellung im aktuellen Internetauftritt „Luftqualität aktuell“

2.1.2 Verkehrsgeräusche aktuell

Zur Dokumentation der aktuellen Situation und der langfristigen Entwicklung von Verkehrsgeräuschen an Verkehrswegen betreibt die LUBW drei Dauermessstation zur Messung des Lärmpegels. Zwei dieser Stationen befinden sich an viel befahrenen Straßen in Karlsruhe und Reutlingen. Die dritte Messstation wurde erst im Frühjahr 2016 in Betrieb genommen und ist an einer Bahntrasse bei Achern positioniert.

Von den Mikrofonen werden Sekundenwerte aggregiert und in eine XML-Datei geschrieben, die einmal in der Minute per FTP auf den zugehörigen Stationsrechner geschoben werden. Dort werden die Daten ausgelesen und in eine relationale Datenbank geschrieben.

Die aktuell gemessenen Schallpegel der letzten halben Stunde werden im Internet als Liniendiagramm dargestellt, welches jede Minute aktualisiert wird (siehe Abb. 2.4).

Des weiteren werden die Stunden-Pegel der letzten 24 Stunden und die Tages- und Nachtpegel der letzten 6 Wochen in Säulendiagrammen dargestellt (vgl. Abb. 2.5).

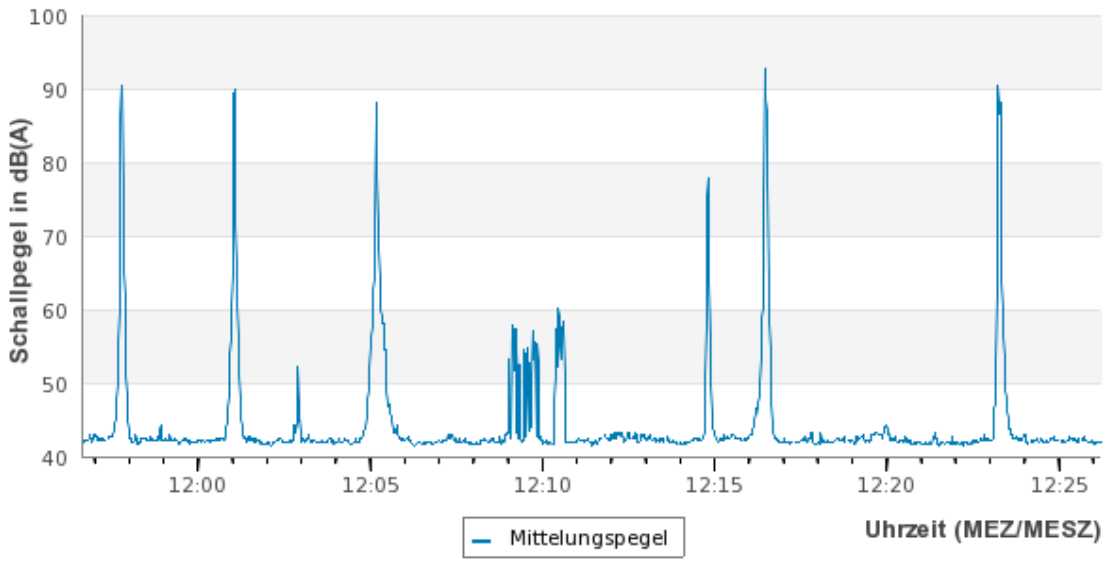

Abbildung 2.4: Liniendiagramm im aktuellen Internetauftritt „Verkehrsgeräusche aktuell“

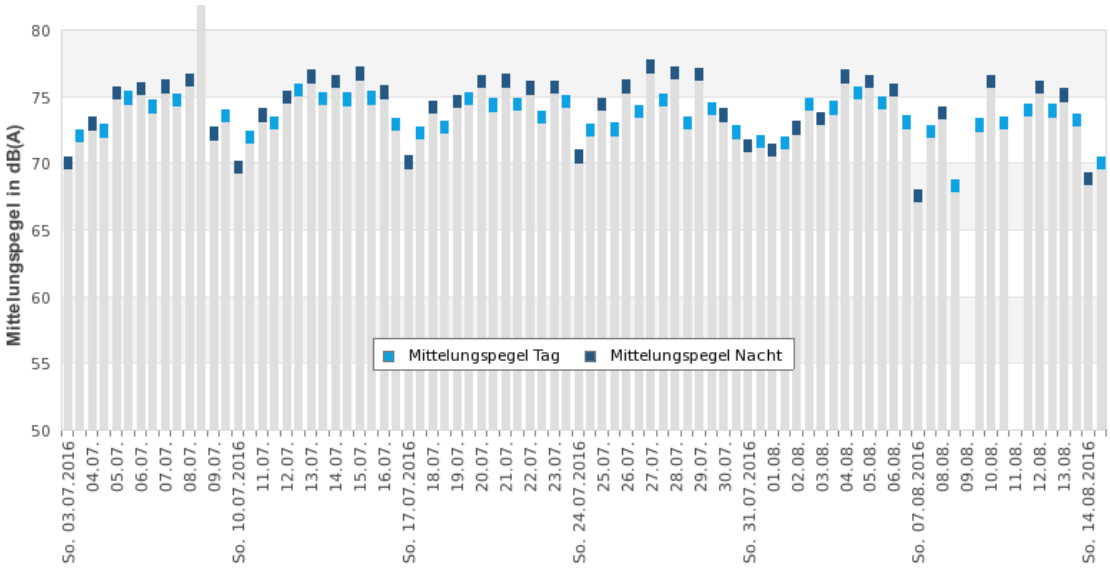

Abbildung 2.5: Säulendiagramm im aktuellen Internetauftritt „Verkehrsgeräusche aktuell“

2.1.3 Randbedingungen

Es wurde bereits damit begonnen, Seiten des LUBW-Webauftritts auf die neue Portaltechnologie Liferay umzustellen. Momentan ist aber noch an vielen Stellen das Content-Management-System (CMS) WebGenesis im Einsatz, bzw. werden die Inhalte auf statischen Hypertext Markup Language (HTML)-Seiten präsentiert. Bei allen drei Varianten ist es möglich, HTML-Quelltext und JavaScript-Code zum bestehenden Seiten-Code hinzuzufügen.

Aus Sicherheitsgründen werden die neuen Liferay-Seiten über Hypertext Transfer Protocol Secure (HTTPS) bereitgestellt. Falls eine solche HTTPS-Seite Inhalte enthält, die über reguläres HTTP abgerufen werden, ist die Verbindung nur teilweise verschlüsselt. Solche Inhalte werden als „Mixed Content“ bezeichnet und gelten im Allgemeinen als unsicher, weshalb sie auch von Browsern standardmäßig nicht angezeigt werden. Aus diesem Grund müssen alle Inhalte (z.B. auch Bibliotheken oder Daten-Objekte), die auf Liferay-Seiten angezeigt werden sollen, ebenfalls über HTTPS-Schnittstellen bereitgestellt werden. Hintergrund dieser Vorgehensweise ist die Sicherstellung der Authentizität der LUBW als Herausgeber dieser amtlichen Messdaten.

2.2 Anforderungen

Die in den folgenden Abschnitten beschriebenen Anforderungen werden in drei Kategorien unterteilt. Dabei wird jede Anforderung eindeutig durch einen Buchstaben und eine Zahl identifiziert, wobei sich der Buchstabe von der Kategorie ableitet, der die Anforderung zugeordnet wurde.

- **Fachseite (F):** Unter diese Kategorie fallen Anforderungen, die von den momentanen Visualisierungsformen erfüllt werden und auch bei einer neuen Umsetzung erfüllt werden sollen.
- **Datenmanager (D):** Diese Anforderungen betreffen Mitarbeiter von der Fachseite, die verantwortlich dafür sind, welche Daten und in welcher Form im Internet veröffentlicht werden.
- **Endnutzer (E):** Dies sind Anforderungen, die interessierte Bürger haben, die sich auf den Seiten der LUBW über ihre Umwelt informieren möchten.

2.2.1 Fachseite

F 1: Vergleichbarkeit von Verlaufsgrafiken

Verlaufsgrafiken sollen möglichst einfach zu vergleichen sein. So sollte man beim Betrachten von unterschiedlichen Grafiken direkt erkennen können, ob sich dabei um eher hohe oder eher niedrige Werte handelt. Dazu muss es für jede Komponente ein festgelegte Skalierung der y-Achse geben.

F 2: Klassifizierung von Werten und entsprechende Kennzeichnung

Messwerte müssen je nach Komponenten nach verschiedenen Kriterien (z.B. Größe oder Tageszeit) klassifiziert werden können. In Visualisierungen müssen diese unterschiedlichen Klassen dann durch entsprechende Einfärbungen gekennzeichnet werden (vgl. Abb. 2.2). Besonders häufig gibt es auch den Fall, dass eine Grenzwertüberschreitung von Komponenten hervorgehoben werden soll.

F 3: Gestapelte Säulen- und Balkendiagramme

Es sollte möglich sein, in einem Säulen- oder Balkendiagramm mehrere zusammengehörende Werte auf einer Säule bzw. einem Balken abbilden zu können (vgl. Abb. 2.5).

F 4: Interaktive Karten-Marker

Über Marker in einer Karte sollten Informationen über die entsprechende Station abrufbar sein. Je nach Endgerät könne Information nach einem Klick auf den Marker oder schon beim Mouseover angezeigt werden. Abbildung 2.6 zeigt, wie so eine Infobox zukünftig aussehen könnte. Hier soll neben dem Namen und den aktuelle Messwerten auch eine Wochengrafik für die entsprechende Komponente angezeigt werden.

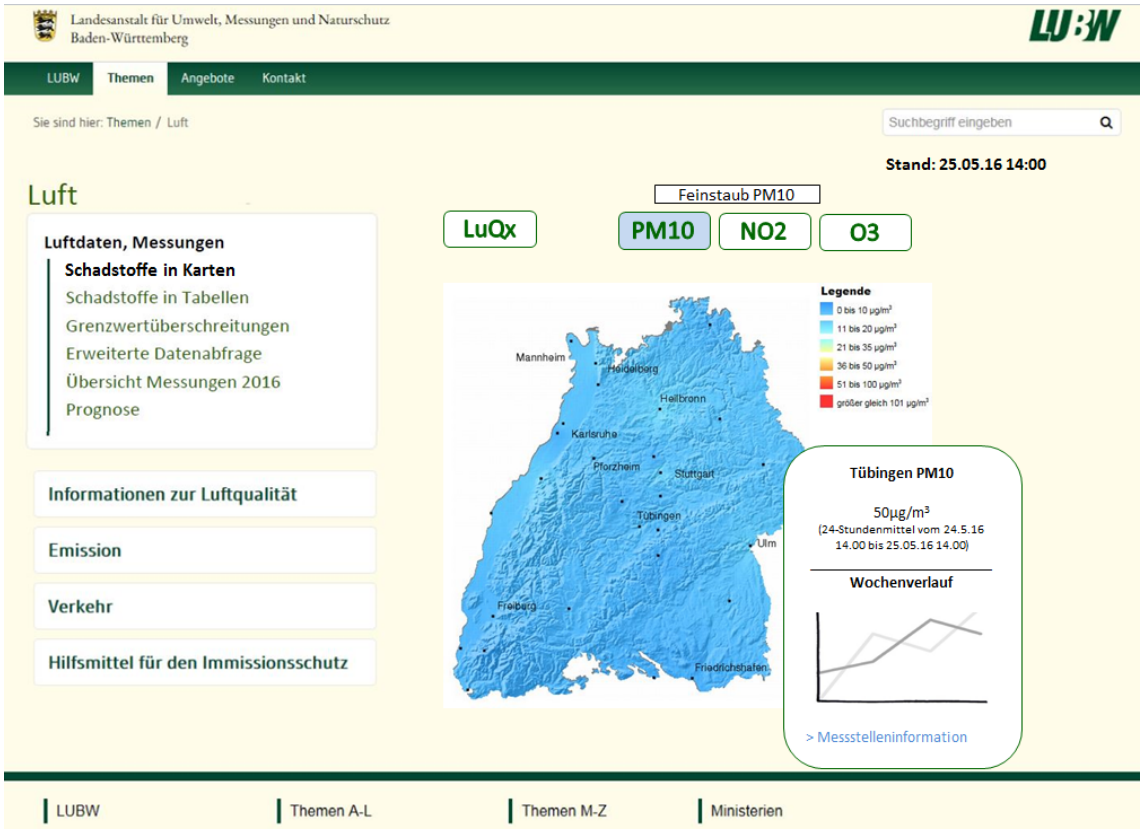

Abbildung 2.6: Mockup: Kartendarstellung auf der neuen Luftseite [24]

F 5: Einzelwerte auf Karte interpolieren

Es sollte möglich, sein die gegebenen Werte einzelner Komponenten über ganz Baden-Württemberg zu interpolieren und Flächen je nach Höhe der berechneten Werte einzufärben (vgl. Abb. 2.3). Bei der Berechnung von Interpolationswerten müssen weitere Faktoren wie beispielsweise Landschaftstopologie oder meteorologische Daten beachtet werden, weshalb die genauen Berechnungsvorschriften von den Fachbereichen festgelegt, sowie die resultierenden raumbezogenen Rasterdaten geliefert werden müssen.

2.2.2 Datenmanager

D 1: Datenerfassung aus verschiedenen Quellen

Je nach Fachbereich werden die Daten in unterschiedlichster Form zur Veröffentlichung im Internet bereitgestellt. Da der Veröffentlichung der Daten auf der Fachseite zumeist komplexe Workflows zur Plausibilisierung der Daten vorgeschaltet sind, sollte dieser Ablauf durch die Visualisierung im Internet möglichst nicht beeinträchtigt werden. D.h. es müssen für alle Formen, in denen Daten vorliegen können, passende Schnittstellen implementiert werden. Bei der bisherigen Analyse wurden bisher folgende Quellsysteme identifiziert: XML-Dateien, CSV-Dateien, SQL-Datenbank (PostgreSql), REST-API (Json).

D 2: Datenexport in unterschiedliche Zielsysteme

Je nachdem wie die Daten weiterverwendet werden, sollten sie in einem möglichst passenden Speicher abgelegt werden. Bei der bisherigen Analyse wurden bisher folgende Zielsysteme identifiziert: SQL-Datenbank (CartoDB), NoSQL (z.B. Elasticsearch), Time-Series-DB (z.B. OpenTSDB).

D 3: Daten-Aggregation

Die Daten, die im Internet dargestellt werden sollen, müssen in manchen Fällen erst aus den Rohdaten aggregiert werden. Dabei handelt es sich zumeist um Mittelwerte über bestimmte Zeitspannen (z.B. Tages- und Nachtpegel auf den Lärm-Seiten). Die Aggregation von Mittelwerten erfolgt bei freigegebenen Daten durch die Fachbereiche. Bei Rohdaten ohne Gewährleistung erfolgt die Aggregation teilweise auch im Webcache.

D 4: Daten-Plausibilisierung

Im Internet veröffentlichte Werte können auf unplausibilisierten Rohdaten basieren, z.B. bei Verkehrslärm. Bei der Aggregation weiterer Werte wird in diesem Fall jedoch eine automatische Plausibilisierung anhand von meteorologischen Daten vorgenommen. Sollte ein festgelegter Anteil von Messwerten, die zur Berechnung eines Mittelwertes herangezogen werden, ungültig sein, werden diese nicht im Internet veröffentlicht.

D 5: Einfache Erzeugbarkeit und Editierbarkeit von Darstellungsformen

Es sollte möglich sein, ohne detaillierte technische Kenntnisse über die verwendeten Internettechnologien, Darstellungen über ein Konfigurationsmenü erstellen und editieren zu können.

D 6: Mehrfachverwendung von Daten

Es werden generische Schnittstellen benötigt, um eine Nutzung der Daten für unterschiedliche Kontexte zu erreichen (Apps, Fachportale und allgemeine Webangebote).

D 7: Update von Daten

Es können Messstationen hinzukommen, wegfallen oder ihren Standort wechseln. Außerdem kann es vorkommen, dass Messreihen nachträglich als ungültig deklariert oder korrigiert werden. Solche Änderungen an den Urdaten sollten möglichst automatisiert erkannt und auch für die im Internet veröffentlichten Daten übernommen werden.

2.2.3 Endnutzer

E 1: Responsivität

Ein Internetnutzer erwartet, dass sich die Darstellung aller Elemente an die (technischen) Möglichkeiten des jeweiligen Endgerätes (PC, Laptop, Tablet, Smartphone) anpasst.

E 2: Automatisches Aktualisieren

Neue Daten werden in unterschiedlichen Intervallen oder gar ohne konkrete zeitliche Abhängigkeiten von den Fachbereichen bereitgestellt. Die unterschiedlichen Darstellungen in unterschiedlichen Endanwendungen (App, Portal, Webseiten) sollten sich alle aktualisieren, sobald neue Daten vorliegen, um so einen konsistente Sicht auf dieselben Daten zu gewährleisten, d.h. die Werte einer Komponente müssen über alle Kommunikationsmedien hinweg identisch sein.

Bei Diagrammen, die sich häufig aktualisieren, sollte das automatische Aktualisieren abschaltbar sein. Bei Verlaufsgraphiken von Momentanwerten sollte außerdem nicht immer das komplette Diagramm oder gar die ganze Seite neu geladen werden. Vielmehr sollten sich die alten Werte nach links verschieben und das Diagramm nur für die neuen Werte neu gezeichnet werden.

E 3: Hineinzoomen in Diagramme

Gerade bei Verlaufsgrafiken erwarten Anwender heutzutage ein gewisse Maß an Interaktivität, beispielsweise dass man in einen bestimmten Zeitbereich hineinzoomen, oder über einen Slider Start- und Endpunkt einer Messreihe auswählen kann. Dabei sollte sich die Achsen-Skalierung automatisch anpassen und sich gegebenenfalls auch die Messwerte-Auflösung ändert.

E 4: Interaktive Infoboxen in Diagrammen

Beim Mouseover (oder Klicken) über eine Linie, einen Balken oder eine Säule sollte eine Info-Fenster mit den konkreten Werten zu diesem Zeitpunkt erscheinen. Des weiteren könnten auch Zeitbereiche, in denen besondere Ereignisse (z.B. Zugvorbeifahrt oder Überschreitung eines Grenzwertes) eingetreten sind, im Diagramm farblich gekennzeichnet werden. Denkbar wäre auch die Anzeige eines Markers. Beim Mouseover oder Klicken auf solche Marker könnten dann weitere Informationen angezeigt werden.

E 5: Gegenüberstellung von Messreihen

Anzeige von Verlaufsgrafiken einer Komponente aus mehreren, frei wählbaren Stationen in einem Diagramm. Damit solche Diagramme nicht zu unübersichtlich werden, sollte die Anzahl gleichzeitig in einem Diagramm anzeigbaren Verlaufsgrafiken auf maximal 5 beschränkt werden.

E 6: Interaktive Sortierung von Tabellen

Die Einträge in Tabellen sollten nach Werten in verschiedenen Spalten umsortierbar sein. Beispielsweise alphabetisch nach Namen oder nach der Höhe von numerischen Werten.

E 7: Hineinzoomen in Karten

Bei Karten ist man es heutzutage gewohnt, dass man in beliebige Bereich hineinzoomen kann und in dem entsprechenden Bereich dann auch mehr Details angezeigt bekommt.

E 8: Ein-/Ausschalten von Kartenelementen

Bei vielen Karten kann man heutzutage Elemente (Marker, Kartenlayer), die in einer Karte angezeigt werden, über Checkboxen beliebig an- und ausschalten.

2.3 Zusammenfassung und konkrete Zielsetzung

Die MNZ verwaltet sehr viele Daten und die bestehenden Strukturen sind im Laufe einiger Jahre entstanden. Hier muss erst abgestimmt werden, welche Daten in welchem Format im neuen Internetauftritt dargestellt werden sollen.

Beim Fachbereich Lärm werden momentan Messwerte von nur drei Stationen im Internet dargestellt. Der Datenfluss ist vergleichsweise einfach. Außerdem wurde er komplett von einer LUBW-internen Arbeitsgruppe umgesetzt und ist gut dokumentiert. Für die Schienenlärmmessung wurde neben einer produktiv eingesetzten MERSY-Station bereits eine Test-Station eingerichtet, die parallel mit den gleichen Daten versorgt wird.

Den gesamten Umfang an möglichen Datenvisualisierungen genauer zu betrachten würde den Rahmen dieser Arbeit sprengen. Es werden deshalb hauptsächlich Visualisierungen der Daten in Diagrammen betrachtet. Dementsprechend sind auch nur die Endnutzer-Anforderungen E 1 - E 5 und die Fach-Anforderungen F 1 - F 3 für eine spätere Beurteilung der umgesetzten Prototypen relevant.

Die Anforderungen eines Datenmanagers sind im wesentlichen unabhängig von der Darstellungsform und müssen deshalb alle bei der Konzeption beachtet werden. D 1 und D 2 sind dabei die grundlegenden Anforderungen, die von der entworfen Architektur auf jeden Fall abgedeckt werden müssen. Die Anforderungen an Datenaggregation (D 3) und –plausibilisierung (D 4) sind jedoch bei einer produktiven Umsetzung erst in einer nachfolgenden Version vorgesehen, weshalb sie auch bei einer prototypischen Umsetzung als eher nachrangig betrachtet werden können.

Die folgende Technologieauswahl bzw. das zu erstellten Konzept berücksichtigen aber alle beschriebenen Anforderungen.

3 Technologie-Analyse

In diesem Kapitel werden moderne informationstechnische Technologien dahingehend untersucht, inwieweit sie die in Abschnitt 2.2 definierten Anforderungen erfüllen können.

3.1 Datenhaltung

Um große Datenmengen effizient, widerspruchsfrei und dauerhaft abspeichern zu können, verwendet man Datenbanken. Es gibt verschiedene Arten von Datenbanken, welche sich durch ihr zugrundeliegendes Datenbankmodell unterscheiden. Dieses definiert, wie Daten gespeichert und verwaltet werden.

Bei den meisten heute existierenden Datenbanken handelt sich um relationale Datenbanken. Seit einigen Jahren gewinnen jedoch weitere Klassen von Datenbanken (z.B. Graphdatenbanken, Key-Value-Stores, Objektdatenbanken, spaltenorientierte DBs, Dokument-DBs) immer mehr an Bedeutung. Im Allgemeinen werden solche Datenbanken häufig unter dem Namen NoSQL-Datenbanken zusammengefasst.

In den folgenden Abschnitten werden die wichtigsten Prinzipien der Datenhaltung vorgestellt, die man bei der Konzeption einer Architektur zur Bereitstellung von Messdaten beachten sollte.

3.1.1 CAP-Theorem

CAP ist eine Abkürzung der drei Begriffe Consistency (Konsistenz), Availability (Verfügbarkeit) und Partition Tolerance (Ausfalltoleranz).

Im Jahr 2000 stellte Eric Brewer das Theorem auf, dass eine verteilte Datenbank immer maximal nur zwei der drei folgenden Eigenschaften garantieren kann:

1. Consistency: Die Datenbank stellt sicher, dass bei zwei gleichzeitigen Leseoperationen auf unterschiedliche Knoten immer dasselbe Ergebnis geliefert wird.
2. Availability: Das System ist immer verfügbar und die Antwortzeiten - vor allem von Leseoperationen - liegen immer unterhalb eines festgelegten Grenzwertes.

3. Partition Toleranz: Das gesamte System darf nicht ausfallen, auch wenn Hardware-Komponenten und/oder Kommunikationsverbindungen ausfallen.

Seth Gilbert und Nancy Lynch konnten dieses Theorem 2002 beweisen. Bei der Entwicklung einer Anwendung muss man sich zunächst also immer entscheiden, welche zwei dieser drei Eigenschaften am wichtigsten sind und dementsprechend seine Datenbank wählen.

Bei relationalen Datenbanken liegt der Schwerpunkt dabei immer auf der Konsistenz. Implementiert wird diese dabei durch das relationale Transaktionskonzept und die Wahrung der ACID-Eigenschaften.

Bei NoSQL-Systemen ist eine Forderung nach Konsistenz eher nebensächlich. Hier sind hohe Verfügbarkeit und Skalierbarkeit das primäre Ziel, weshalb man hierfür das optimistischere Konsistenzmodell BASE entwickelt hat (vgl. Anschnitt 3.1.3). [19, S. 2 ff] [7]

3.1.2 Relationale Datenbanken

Die Daten in einer relationalen Datenbank werden in Relationen gespeichert. Diese kann man sich im Grunde als Tabellen vorstellen, die eine logisch zusammenhängende Einheit von Informationen bilden. Sie bestehen aus einer festen Anzahl von Attributen (Spalten), sowie einer variablen Anzahl an Tupeln (Zeilen), wobei eine Reihenfolge sowohl für Tupel als auch für Attribute dabei nicht definiert ist. [29, S. 30]

Jeder Eintrag (Tupel) innerhalb einer Relation kann über einen sogenannten Primärschlüssel identifiziert werden. Über Fremdschlüssel werden Verbindungen zwischen Relationen hergestellt. Abhängig von Redundanzen und Zugriffsproblemen werden Relationen sogenannten Normalformen zugeordnet. [30, S. 55 ff]

Hauptsächliche Gründe, warum relationale Datenbanken so erfolgreich sind und so häufig eingesetzt werden, sind zum einen die sehr mächtige Abfragesprache Structured Query Language (SQL) und zum anderen die Garantie der entsprechenden Datenbankmanagementsysteme (DBMS), dass bei jedem Verarbeitungsschritt die ACID-Eigenschaften erfüllt werden.

Bei der Verarbeitung und Analyse von großen und ständig immer weiter wachsenden Datenvolumina stoßen herkömmliche relationale Datenbanken jedoch an ihre Grenzen, da eine Skalierung mit zunehmend hohen Kosten verbunden ist. Dies ist der Hauptgrund, warum Unternehmen wie Google oder Twitter eigene Lösungen entwickelten, die sich einfach horizontal skalieren lassen (vgl. Fokus auf horizontale Skalierbarkeit in Abschnitt 3.1.3). [31]

3.1.3 NoSQL-Datenbanken

Diese Art von Datenbank hat in den letzten Jahren stark an Bedeutung gewonnen. Der Begriff NoSQL steht heute als Abkürzung für „Not only SQL“ und bezeichnet eine neue Generation von Datenbanken. Es gibt keine einheitliche Definition. Jedoch berücksichtigen nach Edlich die meisten einige der nachfolgenden Punkte: [11, S. 2 ff]

1. **Kein relationales Datenmodell:** Jede Anforderung kann mit unterschiedlichen Datenmodellen (z.B. auch objektorientiert, graphorientiert oder Key-Value) modelliert werden, wobei nicht immer das relationale Datenbankmodell am besten passt.

2. **Fokus auf horizontale Skalierbarkeit:** Schnelle Reaktionszeiten auf Anfragen und Manipulationen auf immer größer werdende Datenmengen kann man nur garantieren, wenn das zugrundeliegende System horizontal skalierbar ist. Bei einer horizontalen Skalierung (scale out) kann man durch Hinzufügen weiterer Rechner einen Leistungszuwachs erreichen. Dabei handelt sich also immer um verteilte Systeme. Im Gegensatz dazu wird beim vertikalen Skalieren (scale up) derselbe Rechner mit besserer Hardware aufgerüstet. Diese Art der Skalierung ist zum einen sehr teuer, sobald man nicht mehr auf massenproduzierte Standardhardware zurückgreifen kann. Zum anderen gelangt man bei ständig steigender Last damit auch irgendwann an die Grenze des technisch Machbaren. [45]

3. **Schemafrei oder nur schwächere Schemarestriktionen:** Schemaerweiterungen in relationalen Datenbanken sind aufwendig durchzuführen. Da aber gerade heutige Web-Projekte agil sein müssen - auch in ihren Datenstrukturen - war es notwendig, Schemarestriktionen immer weiter aufzulockern.

4. **Einfache Datenreplikation:** Dieses Kriterium leitet sich direkt aus der Grundidee ab, den Datenbestand auf viele Knoten zu verteilen, damit Daten auch bei Ausfällen von Knoten noch verfügbar bleiben.

5. **Einfache API:** Viele NoSQL-Systme bieten eine RESTful API’s, die einfacher als SQL, jedoch oft weniger mächtig sind.

6. **Anderes Konsistenzmodell als ACID:** In verteilten Systemen ist es sehr kompliziert die ACID-Eigenschaften einzuhalten. Außerdem verhindert eine Umsetzung auch eine effektive horizontale Skalierbarkeit, was Performance-Verluste zur Folge hat. In einer Vielzahl von Internetanwendungen gibt es Daten, bei denen es unproblematisch ist, wenn sie für ein kurzes Zeitfenster inkonsistent sind (Eventually Consistent). Für solche Systeme ist die optimistischere Basically Available, Soft State, Eventually Consistent (BASE)-Anforderung ausreichend. [19]

Die Abkürzung BASE bezeichnet ein optimistisches Konsistenzmodell, welches gänzlich ohne Sperren auskommt. Stattdessen werden konkurrierende Datenzugriffe über das Multiversion Concurrency Control (MVCC)-Verfahren koordiniert. Dabei wird bei jeder Datenmanipulation eine neue Version erzeugt, die auf ihre Vorgängerversion verweist. Nach der Beendigung der Transaktion wird diese Version dann mit der aktuellen Version zusammengeführt. In manchen Fällen kann es dabei zu unlösbaren Konflikten kommen, wenn in der Zwischenzeit ein bearbeitetes Attribut auch von einer anderen Anwendung geändert wurde. In einem solchen Fall wird die Version verworfen und eine Fehlermeldung gesendet. [8]

Konsistenz ist bei BASE demnach ein Zustand, der irgendwann erreicht wird, man spricht deshalb auch von Eventually Consistent. Dabei gilt jedoch zu beachten, dass damit nicht wie bei relationalen Datenbanken die Integrität gemeint ist, sondern allein die Korrektheit des Inhalts der abzuspeichernden Daten. [6]

3.1.4 Time-Series-Datenbanken

Eine Time Series Database (TSDB) ist ein Softwaresystem welches für den Umgang mit Zeitreihen-Daten optimiert ist. Zeitreihen-Daten können zum Beispiel Aktienkurse oder Messwerte von Sensoren im sogenannten „Internet of Things“[1] sein. Bei TSDB's handelt es sich um Key-Value-Datenbanken, wobei der Key ein Zeitstempel und der Value ein numerischer Wert ist. [46]

Grundsätzlich kann man für das Speichern und Abfragen von Zeitreihen auch andere Datenbanktypen verwenden. Bei entsprechenden Anwendungen mit typischerweise hohen Transaktionsvolumina, bedarf es jedoch Systemen, die auf die effiziente Handhabung solcher extrem homogener Datensätze spezialisiert sind. Die meisten dieser Implementierungen bieten außerdem eine weitere Zugriffsschicht auf die Daten, welche bereits eine Vielzahl von Funktionen für die häufigsten Datenaggregationen „out of the box“ bereitstellt und über die Routinen konfiguriert werden können, wie beispielsweise mit fehlenden Daten bei Ausfällen von Sensoren umgegangen werden soll. [36]

3.1.5 Datums- und Zeitformate nach ISO 8601

Es existieren viele regional unterschiedliche Datums- und Zeitformate, weshalb es beim Austausch von zeitbehafteten Daten häufig Fehler entstehen. Aus diesem Grund hat man

[1] Mit diesem Begriff wird die Vernetzung von Gegenständen mit dem Internet bezeichnet, um über das Internet zu kommunizieren und so verschiedene Aufgaben für den Besitzer übernehmen können (z.B. Informationsversorgung, automatische Bestellungen oder Warnfunktionen). [9]

sich auf den internationalen Standard **ISO 8601** geeinigt. In Deutschland hat sich die Schreibweise im Alltagsgebrauch nicht durchgesetzt.

DBMS erlauben zwar in der Regel eine Datums- und Zeicheneingabe in vielen länderspezifischen Formaten. Dafür muss aber für jedes Format zunächst die richtige Syntax erlernt werden, wodurch wiederum Unstimmigkeiten und somit Fehler auftreten können. Am einfachsten ist es somit, wenn man bei Datenbanken alle Datums- und Zeitangaben immer nach ISO 8601 formatiert.

Im folgenden werden die wichtigsten Regeln, die es zu beachten gilt kurz dargestellt.

Ein vollständiges Datum wird nach ISO 8601 durch eine vierstellige Jahreszahl, sowie eine zweistellige Monats und Tagesangabe dargestellt. Als Trennzeichen kann optional „-“ verwendet werden.

Basisform:	YYYYMMDD	Beispiel:	20160721
Mit Trennzeichen:	YYYY-MM-DD	Beispiel	2016-07-21

Bei der Angabe einer Tageszeit müssen Stunden-, Minuten-, Sekundenwerte immer durch zweistellige Zahlen repräsentiert werden. Als Trennzeichen kann man optional „:“ verwenden. Eine Stunde kann mit Werten von 00 bis 23 angegeben werden. Zeitangaben kann man nach diesem Format beliebig genau als dezimale Bruchteile einer Sekunde angeben, als Trennzeichen sollte „,“ benutzt werden.

Basisform:	hhmmss	Beispiel:	074329
Mit Trennzeichen:	hh:mm:ss	Beispiel	07:43:29
Auf Stunden genau:	hh	Beispiel	22
Auf Millisekunden genau:	hh:mm:ss,f	Beispiel	23:45:21,904

Bei Zeitreihen benötigt man einen Zeitstempel, welcher einen genauen Zeitpunkt kennzeichnet. Ein Zeitstempel besteht im Wesentlichen aus einem Datum und einer Tageszeit, welche entweder durch ein Leerzeichen oder durch „T“ getrennt werden. Optional kann auch die Zeitzone als Differenz zur Koordinierten Weltzeit (UTC) angegeben werden. Wird die Zeitzone nicht angegeben, führt dies häufig zu Unsicherheiten und somit zu Fehlern. Deshalb sollte ein Zeitstempel immer mit Zeitzone versehen werden. Als spezieller Wert kann auch „Z“ für UTC (+00:00) angegeben werden.

UTC:	2016-07-21T06:19:30Z
Mitteleuropäische Zeit Winter (MEZ):	2016-07-21T07:19:30+01:00
Mitteleuropäische Zeit Sommer (MESZ):	2016-07-21T08:19:30+02:00

In sicherheitskritischen Bereichen sollten Zeitstempel immer für die gleiche Zeitzone oder am einfachsten immer in UTC angegeben werden, um so Fehlern vorzubeugen, die durch die Umstellung von Sommer auf Winterzeit entstehen können. [10] [43]

3.2 Data Ingestion

Unter Data Ingestion (dt. Daten-Aufnahme) versteht man einen Prozess, bei dem Daten importiert werden, um sie direkt zu nutzen oder sie in einer Datenbank zu speichern. [38]

Die Daten können dabei in Echtzeit „gestreamed“ oder in Chargen übertragen werden. Beim Streamen werden die Daten unmittelbar, nachdem sie von der Quelle erzeugt wurden importiert. Bei der Datenaufnahme in Chargen werden die Daten in periodischen Zeitintervallen als Pakete importiert.

Gibt es viele Datenquellen mit großem Datenaufkommen und liegen die Daten auch noch in vielen unterschiedlichen Datenformaten vor, ist es eine echte Herausforderung, die Daten mit einer angemessenen Geschwindigkeit aufzunehmen und effizient zu verarbeiten. Aus diesem Grund gibt es eine Vielzahl von sogenannten „Data Ingestion Tools“, die auf bestimmte Plattformen oder Anwendungen zugeschnitten sind.

Solche Tools bieten häufig auch Funktionalitäten zur weiteren Aufbereitung der Daten. Datenaufbereitung (engl. data preparation) umfasst dabei Komponenten wie Reinigung, Validierung, Transformation, sowie das Zusammenführen von Daten aus verschiedenen Quellen (Datenintegration). Solche komplexeren Datenflüsse verwendet man häufig beim Betrieb eines Data-Warehouse. Dort bezeichnet man solche Abläufe dann auch als Extract, Transform, Load (ETL)-Prozesse. [38]

In den folgenden Abschnitten werden einige dieser Tools vorgestellt und analysiert, inwieweit sie geeignet sind Daten aus den unterschiedlichen Datenquellen (vgl. Anforderung D 1 auf S. 10) in unterschiedliche Zielsysteme (vgl. Anforderung D 2 auf S. 10) zu übertragen.

3.2.1 Logstash

Logstash ist eine von der Firma Elastic entwickelte Software, mit der Daten dynamisch aus unterschiedlichen Quellen normalisiert in ein Zielsystem übertragen werden können.

Logstash bildet zusammen mit Elasticsearch und Kibana die Grundkomponenten des „Elastic Stack“, mit dem Daten aus verschiedenen Quellen aufgenommen und in Echtzeit durchsucht, analysiert und visualisiert werden können.

Eine Logstash-Pipeline zur Aufnahme von Daten besteht aus drei Ebenen, einer Input-, einer Filter- und einer Output-Ebene (vgl. Abb. 3.1). Die unterschiedlichen Inputs und Outputs unterstützen dabei sogenannte Codecs, womit man die Datenrepräsentation eines Datenstroms ändern kann. [5]

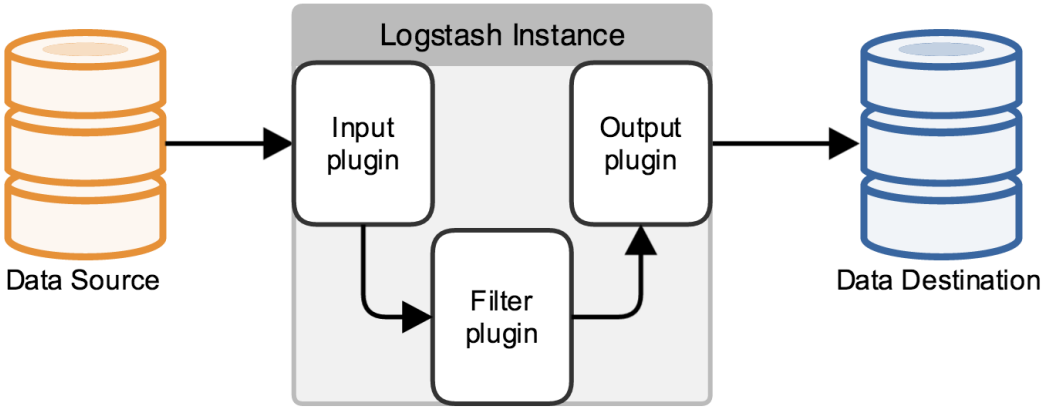

Abbildung 3.1: Aufbau einer Logstash-Pipeline [32]

Ursprünglich wurde Logstash entwickelt, um Informationen aus Logfiles zu filtern und einheitlich ablegen zu können. Die Möglichkeiten gehen aber weit über diesen Anwendungsfall hinaus. Mittlerweile gibt es eine breite Palette von Eingangs-, Filter- und Ausgangs-Plugins, mit denen annähernd jede Art von Ereignis angereicht und transformiert werden kann. [21]

Die momentan existierenden Plugins decken alle Datenquellen ab, die in Anforderung D 1 beschrieben sind und auch alle Zielsysteme aus Anforderung D 2. Zusätzlich gibt es zahlreiche Filter-Plugins mit deren Hilfe man aus allen gängigen Dateiformaten die gewünschten Informationen heraussieben kann. Da es sich um ein Open-Source-Projekt mit einer sehr aktiven Community handelt, kommen ständig neue Plugins für immer speziellere Anforderungen hinzu. Außerdem besteht die Möglichkeit, eigene Plugins zu implementieren.

```
input {
  file {
    path => "/var/log/apache/access.log"
  }
}
filter {
  grok {
      match => { "message" => "%{COMBINEDAPACHELOG}" }
    }

}
```

```
12 output {
13   if [type] == "apache-access" {
14     if "_grokparsefailure" in [tags] {
15       null {}
16     }
17     elasticsearch {
18     }
19   }
20 }
```

Listing 3.1: Beispielkonfiguration von Logstash

Listing 3.1 zeigt eine eine Konfigurationsdatei mit Namen „apache_to_elasticsearch.conf“. Ein Logstash mit dieser Konfiguration detektiert neue Zeilen in der Datei access.log und ließt sie ein. Der Filter parst anschließend den Log-String und speichert alle relevanten Daten, die in Apache-logs enthalten sind. In der Output-Section ab Zeile 14 wird zunächst überprüft, ob die Eingabe dem Pattern eines Apache-logs entsprochen hat. Ist dem so, wird der Event in einem lokalen Elastic Search-Cluster gespeichert, ansonsten wird er verworfen.

3.2.2 Flume

Apache Flume ist ein verteiltes und hochverfügbares System zum effizienten Sammeln, Aggregieren und Bewegen von großen Datenmengen aus verschiedenen Quellen in einen zentralen Datenspeicher wie z.B. Hadoop Distributed File System (HDFS) oder HBase. Wobei es am besten auf ein Hadoop-Ökosystem abgestimmt ist und dort auch häufig zum Einsatz kommt. [13]

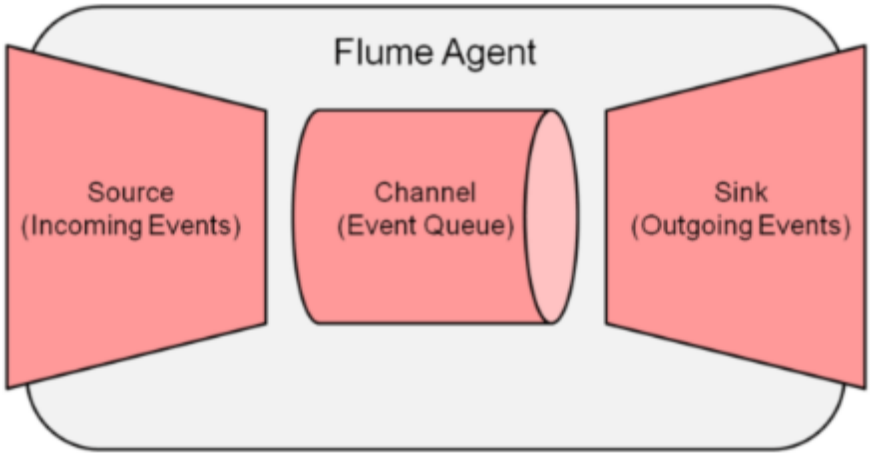

Abbildung 3.2: Aufbau eines Flume Agents [1]

Verschieden Datenflüsse können durch unterschiedliche, sogenannte „Flume Agents“ realisiert werden, von denen jeder auf einer eigene Instanz einer Java Vitual Machine (JVM) läuft. Solche Agenten bestehen aus drei verknüpfbaren Komponenten (vgl. Abb. 3.2):

- **Sources (Quellen):** Flume Sources warten auf Events und nehmen sie entgegen. Auf welche Events genau ein Agent reagiert kann konfiguriert werden. Ein möglicher Event könnte beispielsweise eine neue Ausgabe auf stdout sein oder ein HTTP-Request. Jeder Flume-Agent hat mindestens eine Source, kann aber auch mehrere haben. Wurde von einer Source der entsprechende Event detektiert, schreibt sie ihn in einen Channel per Transaktion.
- **Channels (Kanäle):** Events, die in einen Channel geschrieben werden, bleiben dort, bis sie von einem Sink in einer Transaktion entfernt werden. Auch der Channel kann konfiguriert werden. Man unterscheidet dabei grundsätzlich zwischen in-memory queues für hohen Durchsatz und disk-backed queues für Dauerhaftigkeit.
- **Sinks (Senken):** Durch Sinks wird definiert, wohin die Daten gespeichert werden sollen. Wie bei Sources gibt es auch hier verschiedene Ausgabetypen, die konfiguriert werden können.

Das Schreiben und Lesen auf einen Channel mit Transaktionen, stellt sicher, dass kein Event verloren geht und alle an ihren Bestimmungsort gelangen auch wenn es mal zu Fehlern wie beispielsweise dem Abbruch einer Verbindung kommt. [13]

```
1  # Namen an Komponenten vergeben
2  hdfs-agent.sources= netcat-collect
3  hdfs-agent.sinks = hdfs-write
4  hdfs-agent.channels= memory-channel
5  # Konfiguration der Source
6  hdfs-agent.sources.netcat-collect.type = netcat
7  hdfs-agent.sources.netcat-collect.bind = 127.0.0.1
8  hdfs-agent.sources.netcat-collect.port = 11111
9  # Konfiguration des Sinks
10 hdfs-agent.sinks.hdfs-write.type = hdfs
11 hdfs-agent.sinks.hdfs-write.hdfs.path = hdfs://namenode_address:8020/
       path/to/flume_test
12 hdfs-agent.sinks.hdfs-write.rollInterval = 30
13 hdfs-agent.sinks.hdfs-write.hdfs.writeFormat=Text
14 hdfs-agent.sinks.hdfs-write.hdfs.fileType=DataStream
15 # Konfiguration des Channels
16 hdfs-agent.channels.memoryChannel.type = memory
17 hdfs-agent.channels.memoryChannel.capacity=10000
18 hdfs-agent.sources.netcat-collect.channels=memoryChannel
19 hdfs-agent.sinks.hdfs-write.channel=memoryChannel
```

Listing 3.2: Beispielkonfiguration eines Flume-Agents

Ein Konfigurationsdatei in Flume sieht aus wie eine „Java-properties-Datei“. Listing 3.2 zeigt eine Konfigurationsdatei mit Namen „sample_agent.conf“. Der entsprechende Flume-Agent wartet auf Nachrichten an netcat über Port 11111, puffert sie in einem in-memory channel und speichert sie anschließend in einem HDFS. [1]

3.2.3 FME

FME (Feature Manipulation Engine) ist eine kommerzielle Softwarelösung der Firma Safe mit der komplexe Verarbeitungsprozesse von Daten modelliert und automatisiert ausgeführt werden können. Spezialisiert ist die Software auf räumliche Daten, kann jedoch auch zur Verarbeitung von nicht-räumlichen Daten verwendet werden. FME ist in verschiedenen Versionen (Desktop, Server, Cloud) verfügbar, außerdem gibt es spezielle Versionen für die Arbeit mit Datenbanken oder anderen Programmen wie ESRI ArcGIS.

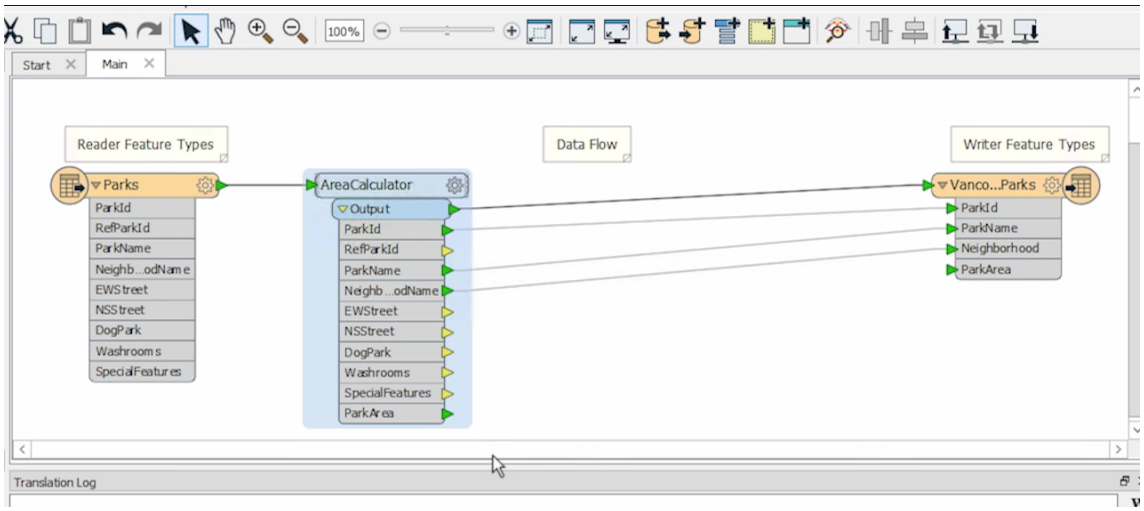

Abbildung 3.3: Aufbau eines FME-Datenflusses mit der Workbench [37]

Über eine graphische Benutzeroberfläche, die FME-Workbench, ist es möglich, Datentransformationen zu erstellen und auszuführen (vgl. Abb. 3.3). Zum Einlesen von Daten benutzt man sogenannte „Reader Feature Types“ und zum schreiben der Daten „Writer Feature Types“. Diese können über ein Konfigurationmenü komfortabel an die entsprechenden Datenquellen angepasst werden. [37]

Vor dem Schreiben in das Zielsystem können die Daten beliebig verändert werden. Hierfür stehen verschiedene Translatoren zur Verfügung, mit denen Geometrie- und Attributdaten bearbeitet werden können. Diese Translatoren lassen sich auf der Workbench beliebig anordnen und so zu benutzerdefinierten Transformationsprozessen verbinden. Zusätzlich

zu den vorhandenen Standard-Translatoren besteht die Möglichkeit auch eigene Python-Skripte zu integrieren. Eine weitere wichtige Komponente ist der Viewer, mit dem erstellte Daten und Zwischenergebnisse betrachtet werden können. [15]

Die Feature-Types bieten eine Anbindung an viele Anwendungen und Webservices und unterstützen das Schreiben und Lesen in alle gängigen Formate. Nach Herstellerangaben kann mit FME ein Import und Export aus über 345 verschiedenen Datenquellen realisiert werden. Auch die in Anforderung D 1 und Anforderung D 2 aufgeführten Datenquellen und Zielsysteme werden damit abgedeckt.

3.3 Visualisierung im Web

In diesem Kapitel werden Technologien behandelt, mit denen die Ansprüche heutiger Internetnutzer erfüllt werden können (vgl. Abschnitt 2.2.3).

Auf dem Markt gibt es eine ganze Menge von JavaScript-Bibliotheken und Anwendungen, mit denen man Web-Visualisierungen von Daten erstellen kann.

Basis aller Frameworks ist HTML5, welches Funktionen wie Video, Audio, lokalen Speicher und dynamische 2D- und 3D-Grafike bietet, die sich zuvor nur mit zusätzlichen Pulgins umsetzen ließen. [42]

In diesem Abschnitt wird außerdem FlexVis genauer beschrieben, wobei es sich um ein Visualisierungs-Framework handelt, welches momentan am Karlsruher Institut für Technologie (KIT) entwickelt wird.

3.3.1 C3.js

C3.js ist eine kostenlose Open-Source-JavaScript-Diagramm-Bibliothek. Sie basiert wie viele andere Tools zum Erstellen von Diagrammen auf D3.js. Dabei wiederum handelt es sich um eine sehr mächtige Bibliothek, mit deren Hilfe man dynamische Graphiken auf HTML5-Basis erzeugen kann. Es gibt dabei jedoch keine Diagrammvorlagen und es bedarf bereits einer gewissen Einarbeitungszeit um damit einfache Charts erstellen zu können. Aus diesem Grund gibt es aufbauend auf D3.js viele Bibliotheken wie eben auch C3.js, die grundlegende Funktionalitäten zur Datenvisualisierung einfacher und konfigurierbar zu Verfügung stellen.

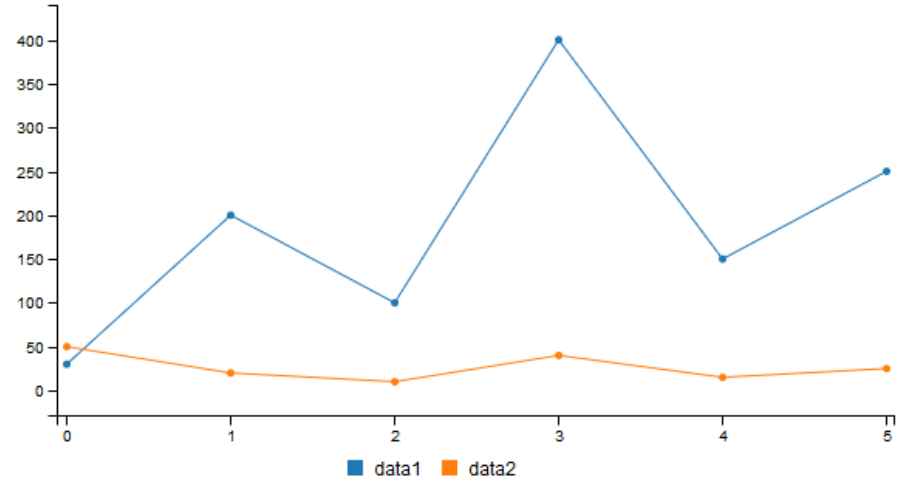

Abbildung 3.4: Liniendiagramm mit C3.js

Wie wenig Code nötig ist, um damit ein einfaches Liniendiagramm zu erzeugen zeigt Listing 3.3. Abbildung 3.4 zeigt das Ergebnis.

```
<!DOCTYPE html>
<html>
<head>
    <title>C3</title>
    <meta charset="utf-8" />
    <link href="c3-0.4.11/c3.css" rel="stylesheet" />
    <script src="d3\d3.js" charset="utf-8"></script>
    <script src="c3-0.4.11/c3.js"></script>
</head>
<body>
    <div id="chart"></div>
    <script>
        var chart = c3.generate({
            data: {
                columns: [
                    ['data1', 30, 200, 100, 400, 150, 250],
                    ['data2', 50, 20, 10, 40, 15, 25]
                ],
                type: 'line'
            }
        });
    </script>
</body>
</html>
```

Listing 3.3: Liniendiagramm mit C3.js [4]

C3.js bietet Schnittellen, über die es möglich ist ein Diagramm zu aktualisieren (Anforderung E 2). [33]

Die Diagramme passen sich dabei an die Bildschirmgröße an und erfüllen somit die Anforderung E 1 (Reponsivität).

3.3.2 Highcharts

Highcharts ist eine bekannte JavaScript-Bibliothek basierend auf HTML5, Scalable Vector Graphics (SVG) und Vector Markup Language (VML). Das Produkt ist seit 2009 auf dem Markt und wird von einer Firma namens Highsoft Solutions AS aus Norwegen entwickelt.

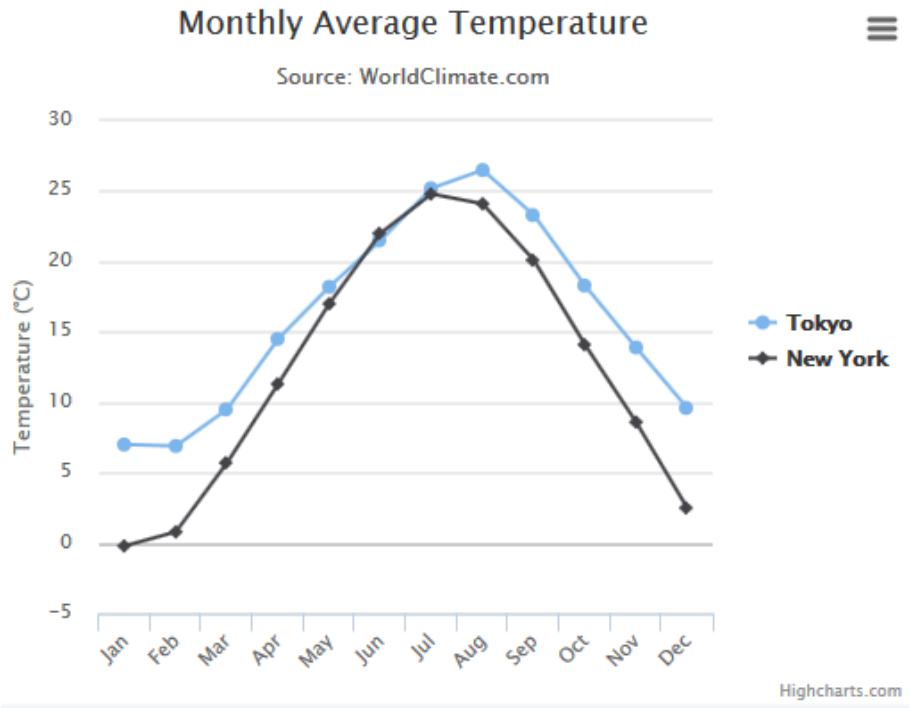

Abbildung 3.5: Liniendiagramm mit Highcharts

Mit Highcharts können alle grundlegenden 2D-Diagramme auf einfache Art erstellt werden. Die Diagramme sehen dabei ansprechend und professionell aus. [20]

Für nicht kommerzielle Zwecke ist Highcharts komplett kostenlos. Für kommerzielle Zwecke gibt es unterschiedliche Lizenzen. [17]

Listing 3.4 zeigt den JavaScript-Code für eine einfaches Liniendiagramm. Abbildung 3.5 zeigt das Ergebnis.

```
1
2 $(function () {
3     $('#container').highcharts({
```

```
        title: {
            text: 'Monthly Average Temperature',
            x: -20 //center
        },
        subtitle: {
            text: 'Source: WorldClimate.com',
            x: -20
        },
        xAxis: {
            categories: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
        },
        yAxis: {
            title: {
                text: 'Temperature (C)'
            },
            plotLines: [{
                value: 0,
                width: 1,
                color: '#808080'
            }]
        },
        tooltip: {
            valueSuffix: 'C'
        },
        legend: {
            layout: 'vertical',
            align: 'right',
            verticalAlign: 'middle',
            borderWidth: 0
        },
        series: [{
            name: 'Tokyo',
            data: [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3,
                18.3, 13.9, 9.6]
        }, {
            name: 'New York',
            data: [-0.2, 0.8, 5.7, 11.3, 17.0, 22.0, 24.8, 24.1, 20.1,
                14.1, 8.6, 2.5]
        }]
    });
});
```

Listing 3.4: Liniendiagramm mit Highcharts [2]

Auf der Homepage von Highcharts[1] findet man viele Diagrammvorlagen, die man direkt online bearbeiten kann und die alle Diagrammformen abdecken, die auf den analysierten Webseiten identifiziert wurden (Anforderungen F 1 - F 3).

Die Diagramme passen sich dabei an die Bildschirmgröße an und sind interaktiv (Anforderungen E 1,E 3 und E 4). Die Diagramme können über geeignete Schnittstellen auch nachträglich aktualisiert werden (Anforderung E 2).

3.3.3 amCharts

amCharts ist der Name einer kleinen Software Firma aus Litauen. Diese entwickelt eine gleichnamige JavaScript-Bibliothek zur Erstellung von 2D-Diagrammen.

Diese Bibliothek kann kostenlos verwendet werden, mit der einzigen Einschränkung, dass in der oberen linken Ecke der Diagramme das Firmenlogo angezeigt wird. Um dies zu unterbinden benötigt man eine kostenpflichtige Lizenz, hat dann aber auch Anspruch auf Support. [26]

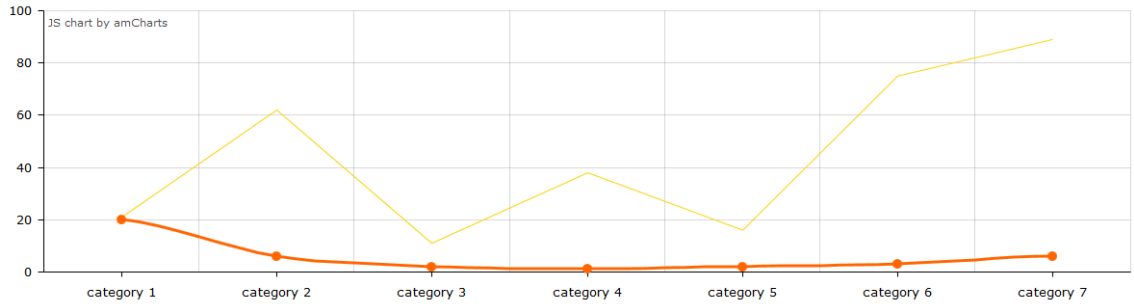

Abbildung 3.6: Liniendiagramm mit amCharts.js

Listing 3.5 zeigt den JavaScript-Code für eine einfaches Liniendiagramm. Abbildung 3.6 zeigt das Ergebnis.

```
1  AmCharts.makeChart("chartdiv",
2  {
3    "type": "serial",
4    "categoryField": "category",
5    "autoMarginOffset": 40,
6    "marginRight": 60,
7    "marginTop": 60,
8    "startDuration": 1,
9    "fontSize": 13,
10   "theme": "default",
```

[1] http://www.highcharts.com/demo

```
11    "categoryAxis": {
12      "gridPosition": "start"
13    },
14    "trendLines": [],
15    "graphs": [
16      {
17        "balloonText": "[[title]] of [[category]]:[[value]]",
18        "bullet": "round",
19        "bulletSize": 10,
20        "id": "AmGraph-1",
21        "lineThickness": 3,
22        "title": "graph 1",
23        "type": "smoothedLine",
24        "valueField": "column-1"
25      },
26      {
27        "gapPeriod": 0,
28        "id": "AmGraph-2",
29        "tabIndex": 2,
30        "title": "graph 2",
31        "valueField": "column-2"
32      }
33    ],
34    "guides": [],
35    "valueAxes": [
36      {
37        "id": "ValueAxis-1",
38        "title": ""
39      }
40    ],
41    "allLabels": [],
42    "balloon": {},
43    "titles": [],
44    "dataProvider": [
45      {
46        "category": "category 1",
47        "column-1": 20,
48        "column-2": 21
49      },
50      //...
51    ]
52  });
```

Listing 3.5: Liniendiagramm mit amCharts

Auf der Homepage von amCharts[1] findet man sehr viele Diagrammvorlagen, deren Code man direkt online bearbeiten kann. Außerdem steht ein Online-Editor zur Verfügung,

mit dem man ein Diagramm auch ohne Programmierkenntnisse über eine GUI erstellen kann. Die vorhandene Vorlagen decken alle Diagramme der analysierten Webseiten ab (Anforderungen F 1 - F 3).

Die Diagramme wirken sehr verspielt und es gibt nahezu unzählige Möglichkeiten, sie nach den eigenen Wünschen anzupassen. Sie sind interaktiv und passen sich der Bildschirmgröße des Endgeräts an (Anforderungen E 1,E 3 und E 4).

Es gibt Schnittstellen, über die Diagramme zur Laufzeit aktualisiert werden können (Anforderung E 2).

3.3.4 FlexVis

FlexVis ist ein generisches Visualisierungs-Framework, welches von Eric Braun im Rahmen seiner Master-Thesis beim Institut für Angewandte Informatik des KIT entworfen wurde und seitdem ständig weiterentwickelt wird.

Ziel der Arbeit war es, eine Anwendung zu entwickeln, mit deren Hilfe Redakteure ohne Programmierkenntnisse unterschiedliche Visualisierungen ihrer Daten über ein Graphical User Interface (GUI) konfigurieren können. [12, S. 2]

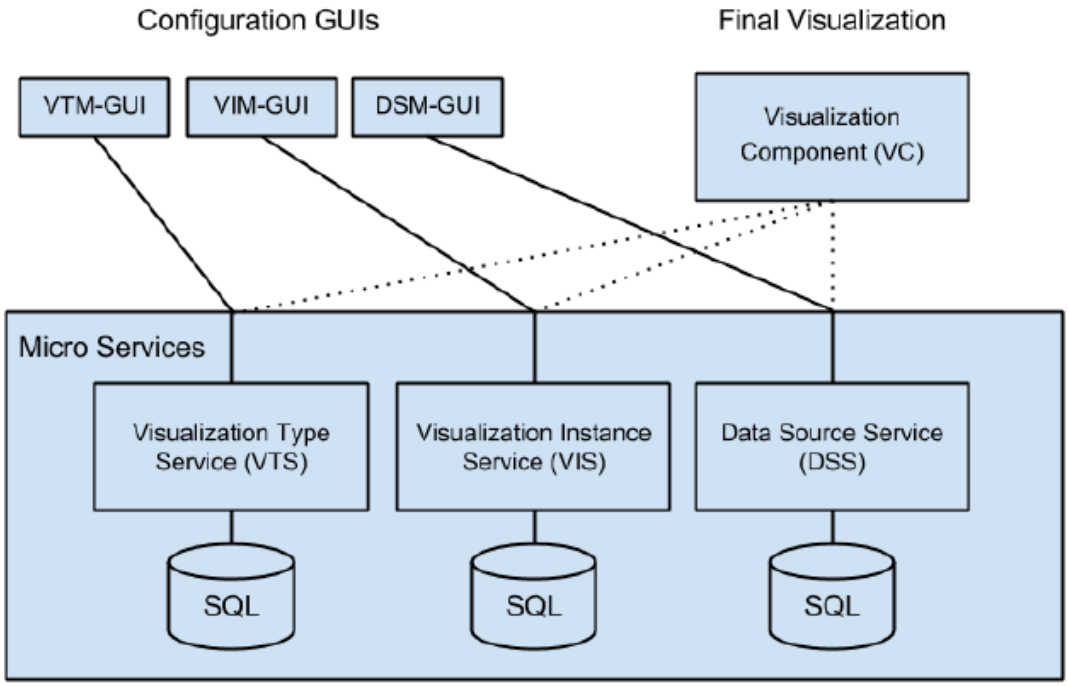

Abbildung 3.7: Architektur von FlexVis [12, S. 35]

[1] https://www.amcharts.com/demos/

Abbildung 3.7 zeigt den grundsätzlichen Aufbau des Frameworks. Wie man erkennen kann, wird die Funktionalität aufgeteilt auf drei unterschiedliche Microservices (vgl. Abschnitt 3.4.2).

Über den Data Source Service (DSS) werden die unterschiedlichen Datenquellen verwaltet und bereitgestellt. Datenquelle kann eine Datei, eine Datenbank oder eine REST-API sein. Von einem Datenmanager können über die DSM-GUI (siehe Abb. 5.5 auf Seite 52) die unterschiedlichen Datenquellen eingerichtet und gepflegt werden. Es ist hierbei möglich, Parameter zu definieren, über welche die Datenquelle auch zur Laufzeit angepasst werden kann.

Der Visualization Type Service (VTS) ist zuständig für die Bereitstellung und Administration des Quellcodes für die unterschiedlichen Visualisierungsformen. Damit der Quellcode eines Visualisierungstyps in das Framework integriert werden kann, muss ein Entwickler darauf achten, dass der Code einer bestimmten Struktur entspricht (siehe Listing 3.6). Auch eine Visualisierung kann mit Parametern ausgestattet werden. Durch Änderungen dieser Parameter können die Visualisierungen dann später von einem Redakteur auf einen konkreten Verwendungszweck hin zugeschnitten werden. Visualisierungtypen können über die VTM-GUI (siehe Abb. 5.4 auf Seite 51) eingepflegt und editiert werden.

```
<dom-module id="meine-komponente">
    <template>
    </template>
    <script>
        (function() {
            Polymer({
                is: "meine-komponente",
                visualization_start: function(data, params) {

                },
                visualization_update: function(data, params) {

                }
            });
        })();
    </script>
</dom-module>
```

Listing 3.6: Struktur einer FlexVis-Visualisierungskomponente [1]

Durch die Trennung von Daten und ihrer Darstellungsform kann beides vielfältig wiederverwendet werden. Zur Realisierung einer konkreten Visualisierung wird ein weiterer Service verwendet, der beides zusammenführt. Dies erledigt der Visualization Instance

[1] Email von Erik Braun am 28 Januar 2016

Service (VIS). Über die VIM-GUI (siehe Abb. 5.6 auf Seite 52) kann ein Redakteur eine oder mehrere Datenquellen auswählen, diese mit einem Visualisierungstyp verknüpfen und die Visualisierung gegebenenfalls noch über die zur Verfügung stehenden Parameter anpassen.

Eine fertige Visualisierungskomponente, die zum Beispiel ein Diagramm auf einer Webseite anzeigt, ruft zur Laufzeit dann zunächst über eine ID auf dem VIS-Service die konfigurierten Beziehungen ab. Anschließend werden die entsprechenden Daten und der Code des Visualisierungstyps abgerufen. Diese werden dann zum Schluss - wie vom Redakteur konfiguriert - miteinander verknüpft und für die Parameter die hinterlegten Werte eingesetzt.

3.4 Backend Konzepte und Technologien

Bei der Architektur, die in dieser Arbeit beschriebenen wird (vgl. Kapitel 4), werden Microservices eingesetzt um einen Datenfluss zu realisieren. Diese kommunizieren über Restful API's und werden in Docker-Containern betrieben. Die folgenden Abschnitte sollen eine Überblick über die grundlegenden Konzepte dieser Technologien geben.

3.4.1 REST

Der Archtektur-Stil Representational State Transfer (REST) wurde von Roy Thomas Fielding in seiner Dissertation mit dem Titel „Architectural Styles and the Design of Network-based Software Architectures“ [28] postuliert. Der Fokus von Rest liegt auf Skalierbarkeit, Erweiterbarkeit und Interoperabilität. Wobei Fielding nur abstrakte Bedingungen einfordert und keine expliziten Prozesse, Protokolle oder gar Medien. Jedoch wird als ein wichtiger Designaspekt die sinnvolle Einbettung erfolgreicher Technologien aufgeführt, weshalb als Beispiel für ein RESTful Design häufig Hypertext Transfer Protocol (HTTP) als Kommunikationsprotokoll herangezogen wird.

Damit ein Dienst als RESTful bezeichnet werden kann, sollte er die folgenden sechs Eigenschaften aufweisen. [22] [44]

- **Client-Server:**

  Strikte Kapselung von Server und Client. Ein Server stellt einen Dienst bereit, der vom Client bei Bedarf angefragt werden kann.

- **Stateless**

  Jede Nachricht muss abgeschlossen sein. D.h. sie muss alle nötigen Informationen beinhalten und nicht abhängig sein von einem Zustand, der auf Client oder Server vorgehalten werden muss.

- **Caching**

  Diese Eigenschaft fordert den effizienten Einsatz von HTTP-Caching, um unnötige Datenübertragungen zu vermeiden und somit Zugriffszeiten zu verringern.

- **Uniform Interface**

  Dies ist die wichtigste Eigenschaft einer REST-Architektur, die sie von anderen Architekturstilen am meisten unterscheidet. Um einheitliche Schnittstellen zu garantieren, wird die Einhaltung der folgenden vier Eigenschaften gefordert:

  - **Identification of resources:** Jede Ressource (Dienst, Information) hat eine eindeutige Adresse.
  - **Manipulation of resources through these representations:** Je nach Anforderung verschiedener Anwendungen können Ressourcen in unterschiedlicher Darstellungsform (Repräsentation) ausgeliefert werden (beispielsweise in einer anderen Sprache oder in einem anderen Format). Als Repräsentationen sollten möglichst vorhandene und verbreitete Media Types verwendet werden.
  - **Self-descriptive messages:** Nachrichten sollen selbstbeschreibend sein. Das bedeutet, dass ausschließlich die Standardmethoden des zugrundeliegenden Protokoll genutzt werden sollen. In den meisten Fällen also die HTTP-Verben (hauptsächlich GET, POST, PUT und DELETE).
  - **Hypermedia as the engine of application state (HATEOAS):** Für Fielding ist dies die wichtigste Eigenschaft einer Restful API, die aber zugleich auch am häufigsten missachtet wird. [27] Sie besagt, dass es für eine Applikation nur eine Einstiegs-Ressource geben sollte. Die Zustandsübergänge, die für einen Benutzer als nächstes möglich sind, werden durch Uniform Resource Identifier (URI)'s dargestellt, die in den unterschiedlichen Ressourcen enthalten sein müssen. Auf diese Weise sollen massiv untereinander verlinkte Ressourcen entstehen, die durch einen Client über Standard-Methoden frei durchlaufen werden können.

- **Layered System:**

  Ein System soll mehrschichtig aufgebaut sein. Ein Anwender sieht jedoch nur die Schnittstelle, wodurch die Architektur insgesamt vereinfacht wird.

- **Code on Demand:**

  Laut Fielding ist diese Eigenschaft optional. Damit ist gemeint, dass Code, der auf dem Client ausgeführt werden soll, erst bei Bedarf nachgeladen wird.

3.4.2 Microservices

Bei einer Microservice-Architektur werden einzelne Anwendungen durch eine Reihe kleiner Services implementiert, die jeweils in einem eigenen Prozess ablaufen und über einfache Schnittstellen (meist HTTP-basierte und RESTful) miteinander kommunizieren. Diese Dienste sind unabhängig von einander deploybar, können also unabhängig von einander entwickelt oder geändert werden. Anstatt, wie bei monolithischen Anwendungen üblich, eine einzige Datenbank für mehrere Dienste, tendiert man bei Microservices eher dazu, jeden Dienst seine eigene Datenbank verwalten zu lassen, um so eine bessere vertikale Trennung der Schichten zu erreichen.

Bei einer Microservice-Architektur ist man nicht anwendungsweit auf bestimmte Technologien (z.B. Programmiersprache, Framework oder Datenbanktechnologie) festgelegt. Vielmehr hat man die Freiheit, erst bei der Implementierung eines bestimmten Features aus unterschiedlichen Technologien die am besten passenden auswählen zu können.[14]

Ein weiterer Vorteil ist, dass man Microservices viel differenzierter skalieren kann. Während man bei monolitischen Anwendungen nur skalieren kann, indem man weitere Kopien des gesamten Systems erzeugt, kann man bei Microservices viel feingranularer nur die Services replizieren, die einen Flaschenhals darstellen (vgl Abb. 3.8).

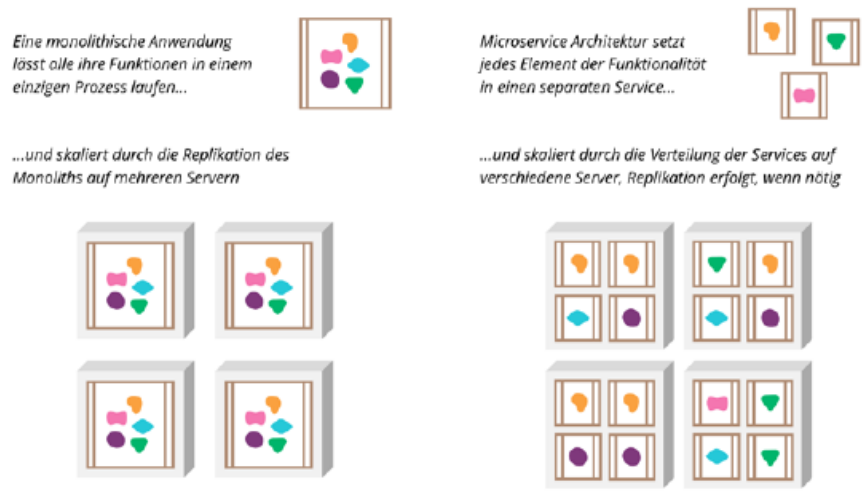

Abbildung 3.8: Vergleich der Skalierbarkeit von Microservices und Monolithen [14, S. 1]

3.4.3 Docker

Docker ist eine Open-Source-Software, die es ermöglicht, Anwendungen und die notwendige Infrastruktur mithilfe von Betriebssystemvirtualisierung in Containern zu isolieren. Diese Container enthalten alle nötigen Pakete und lassen sich leicht als Dateien transportieren und installieren. Des weiteren sind Docker Container komplett unabhängig von der zugrundeliegenden Infrastruktur (Betriebssystem etc.), wodurch Anwendungen flexibel und systemübergreifend entwickelt und einfach bereitgestellt werden können. Sie bilden somit eine optimal Laufzeitumgebung für - per Definition ohnehin gekapselte - Microservices. Da Container außerdem keinen Zugriff auf Ressourcen anderer Container haben, wird eine Trennung der auf einem Rechner genutzten Ressourcen gewährleistet. [41]

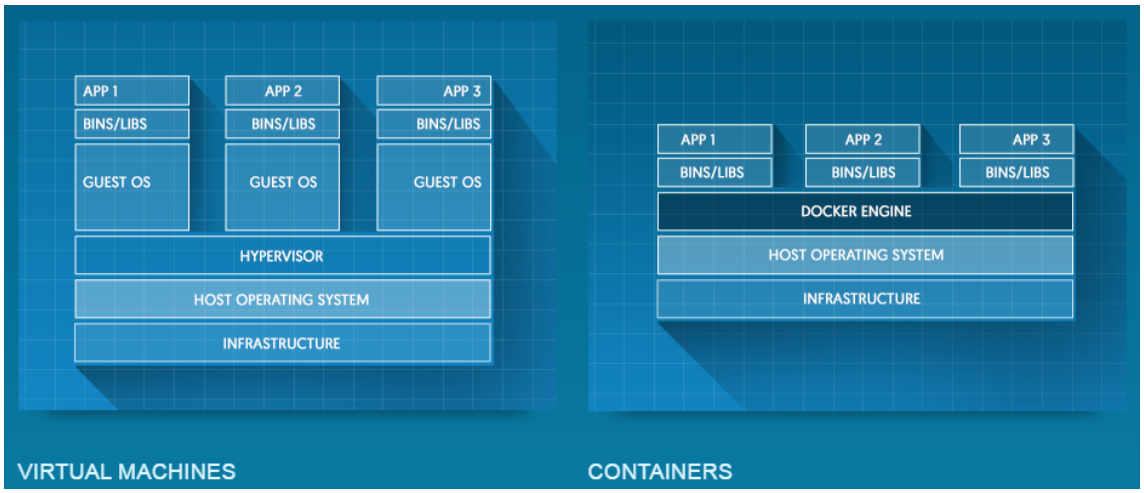

Abbildung 3.9: Vergleich von Docker-Containern und Virtuellen Maschinen [39]

Alle Container die auf dem gleichen Server laufen, teilen sich den Kernel und laufen als isolierte Prozesse auf dem Host Operating System. Es muss also nicht wie bei herkömmlichen Virtuellen Maschinen sämtliche Hardware eines Computer visualisiert werden, weshalb ein Container deutlich weniger Ressourcen benötigt (vgl. Abb. 3.9). [39]

4 Konzeption einer IT-Infrastruktur zur Bereitstellung und Darstellung von Messdaten

In diesem Kapitel wird eine Software-Architektur beschrieben, mit der es möglich ist die Umweltmessdaten der unterschiedlichen Fachbereiche der LUBW im Internet ansprechend und hoch verfügbar zu präsentieren.

Die IT-Welt unterliegt einem permanenten Wandel. So kommen ständig neue innovative Produkte auf den Markt, durch die sich neue Möglichkeiten ergeben. Bei anderen Produkten wird der Support eingestellt oder sie werden gar nicht mehr unterstützt. Aus diesem Grund ist es wichtig, dass heutige Software-Architekturen möglichst modular und generisch aufgebaut sind, damit sie zum einen leicht erweiterbar und zum anderen einzelne Komponenten einfach austauschbar sind. Aus diesen Gründen wurde auch die Gesamtanwendung in mehrere Schichten aufgeteilt.

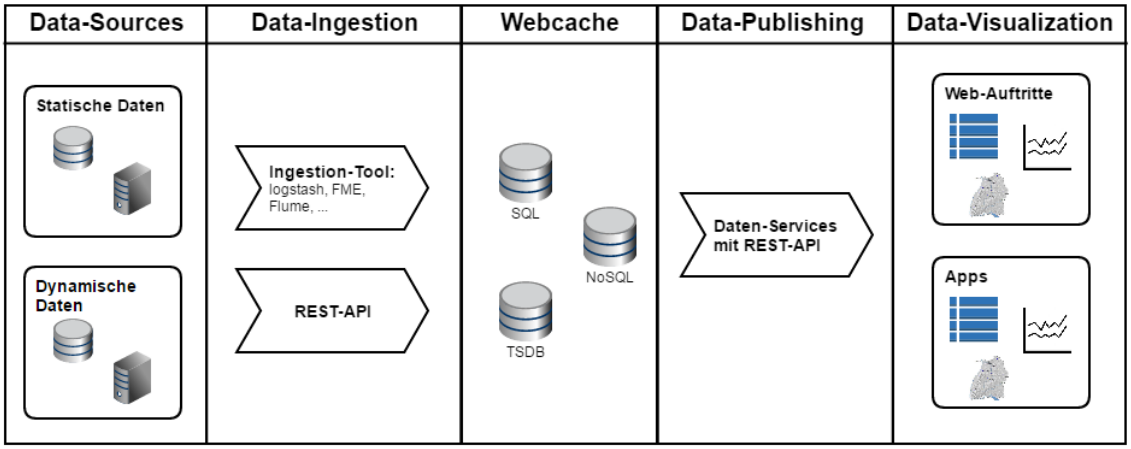

Abbildung 4.1: Architekturmodell

Wie Abbildung 4.1 veranschaulicht, besteht die entworfene Architektur aus fünf Schichten. Ausgehend von den Data-Sources (vgl. Abschnitt 4.1) müssen die Daten automatisiert in den Webcache übertragen werden (vgl. Abschnitt 4.2). Im Webcache (vgl. Abschnitt 4.3)

liegen die Daten hochverfügbar vor und werden über generische Schnittstellen veröffentlicht (vgl. Abschnitt 4.4). Zuletzt werden sie von Visualisierungskomponenten abgefragt und in Diagrammen, Karten oder Tabelle interaktiv und responsiv dargestellt (vgl. Abschnitt 4.5).

Im Folgenden wird zwischen zwei Arten von Daten unterschieden. Zum einen statische Datensätze, bei denen sich die einzelnen Daten gar nicht oder nur selten ändern und deren Umfang annähernd konstant bleibt. Damit sind vor allem Stammdaten von Stationen gemeint.

Zum anderen ist von dynamischen Datensätze die Rede. Damit sind die Messdatenreihen gemeint, auf die häufig schreibend zugegriffen werden muss.

4.1 Data-Sources

Am Anfang jeder Datenverarbeitungskette stehen die Data-Sources. Diese liegen im wesentlichen im Verantwortungsbereich der unterschiedlichen Fachbereiche.

Es müssen jeweils individuelle Absprachen getroffen werden, wie und in welchen Zyklen sie ihre Daten zur weiteren Bearbeitung bereitstellen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Datenaustausch version="69" >
<StaatenBlock StaatName="Deutschland" StaatKenn="D" >
<LaenderBlock LandName="Baden-W&uuml;rttemberg" LandKenn="BW" >
<DatenAusgabe StartDatum="2016-02-08" EndDatum="2016-02-16" >
<Messstelle Name="Stuttgart Am Neckartor" KurzName="S-AN" EUKenn="
    DEBW118" AbrufZeiger="15.02.2016 13:00" >
  <DatenTyp AD-Name="PM10-K-TMW_R" WerteTyp="1" Schwelle="" MessZeit="1D
      " Intervall="1D" >
    <Komponente Name="Schwebstaub PM10" KurzName="PM10" KompKenn=""
        NachweisGrenze="0.1" Einheit="&mu;g/m3" >
      <DatenReihe ZeitPunkt="2016-02-08" >
        <Wert MessEnde="'24:00'" Status=" " >18</Wert>
      </DatenReihe>
      <!--...-->
  </DatenTyp>
<!--...-->
```

Listing 4.1: Beispiel für eine XML-Datei im UBA-Format (Auszug)

Wo es möglich ist, die Datenquelle in ihrem Format zu ändern, sollte möglichst eine einheitliche Lösung angestrebt werden. Eine möglichst einfache Standardlösung könnte

beispielsweise sein, die Daten in einem einheitlichen Format auf einen FTP-Server zu schieben.

Als Austauschformat würde sich ein vom Umweltbundesamt definiertes XML-Format (UBA-Format vgl. Listing 4.1) anbieten, da es gut dokumentiert ist und bereits häufig eingesetzt wird.

Im Hinblick darauf, dass aller Wahrscheinlichkeit nach JavaScript eingesetzt wird, um die Daten im Internet zu visualisieren, wäre es auch denkbar, ein eigenes JSON-Format zu entwickeln.

4.2 Data Ingestion

Je nach den IT-Kenntnissen und Möglichkeiten der unterschiedlichen Fachbereiche sind unterschiedliche Methoden denkbar, wie die Daten in den Webcache gelangen können.

Eine Variante wäre sicherlich eine Schnittstelle zu implementieren, über die die Fachbereiche direkt Daten in den Webcache hinzufügen und manipulieren können. Eine solche Schnittstelle müsste dann sicherstellen, dass jeder Fachbereich nur seine eigenen Daten bearbeiten kann. Die für eine Automatisierung solcher Zugriffe notwendige Routinen und Programme, müssten dann jedoch von den Fachbereichen selbst auf ihren Systemen installiert und implementiert werden. Dazu müsste in den Fachbereichen aber zunächst das nötige Know-How aufgebaut werden.

Ein standardisierte Vorgehensweise ist gewährleistet, wenn alle Übertragungen von Daten in den Webcache in der Verantwortung des ITZ liegen.

Sofern es von den Fachbereichen erlaubt wird, ist es am einfachsten, wenn man in bestimmten Intervallen die neusten Daten aus einer Fachdatenbank ausliest und in den Webcache schreibt.

Ein andere Variante ist die, dass die Fachbereiche ihre Daten in Dateien auf einen FTP-Server pushen. Aus diesen Dateien müssen dann die für eine Veröffentlichung relevanten Daten herausgefiltert und im Webcache einheitlich abgelegt werden.

Für einen Abgleich der Daten auf dem Webcache mit den zugrundeliegenden Daten (vgl. Anforderung D 7) gibt es zwei grundsätzlich verschiedene Ansätze. Zum einen könnte man immer alle Daten übertragen, die im Internet angezeigt werden sollen und die vorherigen Daten damit ersetzen. Mit dieser Variante könnte ein automatischer und zuverlässiger Abgleich einfach implementiert werden. Die Aktualität der Daten ist damit immer gewährleistet.

Eleganter ist es, wenn man nur in den Webcache schreibt, wenn es neue Daten gibt oder sich Daten geändert haben. Da es sich bei Stammdaten um Datensätze von vergleichsweise geringem Umfang handelt und sie sich außerdem nur selten ändern, ist eine derartige Implementierung hierbei einfacher.

So könnten Updates an Stammdaten direkt von den Fachbereichen über eine REST-Schnittstelle durchgeführt werden. Jedoch ist es immer problematisch, wenn zwei Datenbanken mit redundanten Daten manuell abgeglichen werden sollen.

Für einen automatischen Abgleich der Stammdaten, wie in Anforderung D 7 gefordert, könnten man in festgelegten Intervallen (z.B einmal am Tag) mit einem Data-Ingestion-Tool die Datenquelle (direkt die Datenbank oder Dateien auf FTP-Server) auslesen und über eine Schnittstelle an den Webcache weiterleiten. Dass nur Daten geschrieben werden, die neu sind oder sich geändert haben, müsste in der Schnittstelle implementiert werden.

In Fällen, bei denen eine nachträgliche Überprüfung der Messwerte nicht vorgesehen ist, können einfach immer nur die neuen Daten in den Webcache übertragen werden, da sich die alten Daten ja nicht mehr ändern können.

Genauso kann man auch mit den anderen Messreihen verfahren. Es besteht dann jedoch die Möglichkeit, dass im Internet andere Messwerte angezeigt werden wie in der Fachdatenbank stehen. Wo solche Daten vor der endgültigen Abnahme jedoch als vorläufige Daten gelten und auch so im Internet gekennzeichnet werden, kann eine solche Diskrepanz toleriert werden.

Solche vorläufigen Daten sollten nicht länger als nötig gespeichert werden. Sollen ältere Messreihen angezeigt werden, für die es schon endgültige Daten gibt, könnte dafür ein separater Datenfluss implementiert werden, über den die entsprechenden Daten zum Beispiel nur einmal im Monat exportiert werden.

Die drei in Abschnitt 3.2 beschriebenen Lösungen sind generell alle geeignet um Daten aus den unterschiedlichen Quellen in den Webcache zu übertragen. Das Filtern von Informationen aus einer Datei bedarf bei Flume jedoch einiges an zusätzlicher Programmierarbeit, wohingegen es bei Logstash einfach konfiguriert und bei FME über die GUI realisiert werden kann. Des weiteren ist bei Logstash und FME die Auswahl an möglichen Inputs und Outputs viel größer als bei Flume.

Es muss aber nicht zwangsläufig immer das gleiche Data-Ingestion-Tool genutzt werden. Durch den modularen Aufbau der Architektur ist es möglich, für jeden Anwendungsfall das am besten passende Tool zu wählen. So können einfache Datensätze, wie die, welche hier für die prototypische Umsetzung betrachtet wurden, problemlos mit Logstash transferiert werden. In Fällen bei denen aufwendige Datenaggregationen nötigt sind, wie beispielsweise

für dynamische Karten, wird wohl eher FME das Tool der Wahl sein (vgl. Abschnitt 3.2.3).

Für sehr komplexen Datenflüssen, die sehr CPU-intensive Prozesse beinhalten oder die Daten an mehrere Zielsysteme geschickt werden sollen, muss die Architektur gegebenenfalls um Message-Broker ergänzt werden, die das Verteilen und Puffern der Daten übernehmen.

4.3 Webcache

Nach dem CAP-Theorem (vgl. Abschnitt 3.1.1) muss man sich bei der Wahl einer Datenbank entscheiden, welche beiden der drei Eigenschaften Konsistenz, Verfügbarkeit oder Ausfalltoleranz für den jeweiligen Anwendungsfall am wichtigsten sind. Bei der Veröffentlichung von Messdaten im Internet ist es tolerierbar, wenn Änderungen nicht bei allen Usern gleichzeitig eintreffen. Viel wichtiger ist für interessierte Bürger eine permanente Verfügbarkeit und geringe Latenzzeiten. Tendenziell würde man also eher zu einer NoSQL-Datenbank tendieren, da solche Lösungen grundsätzlich auf horizontale Skalierbarkeit ausgelegt sind (vgl. Abschnitt 3.1.3).

Mittlerweile gibt es jedoch auch relationale Datenbanken für Public/Private-Cloud-Umgebungen, die einfache API's über HTTP bieten und außerdem skalierbar sind, um auch im Falle hoher Zugriffszahlen kurze Antwortzeiten zu garantieren. Für Stammdaten von Stationen würden sich solche Datenbanken anbieten, da sie einen eher kleinen und konstanten Umfang aufweisen. Außerdem liegen diese Daten bereits in relationalen Datenbanken in festen Schemata vor, die man für die Webcache-Lösung übernehmen könnte.

Bei den Messwerten handelt es sich um Zeitreihen. Deshalb könnte man dafür eine TSDB verwenden, da solche Datenbanklösungen speziell für diese Art von Daten spezialisiert sind. Durch diese hohe Spezialisierung sind solche Datenbanken aber sehr unflexibel einsetzbar. So kann man meist zu jedem Zeitstempel nur einen numerischen Wert speichern (siehe 3.1.4). Zusätzliche Metainformationen zu beispielsweise Ereignissen wie Züge im Kontext Lärmmessung müssen in einer zweiten Datenbank hinterlegt werden.

Als gute Alternative zur Speicherung solcher Daten hat sich Elasticsearch erwiesen. Bei dieser Software werden alle sechs Eigenschaften berücksichtigt, die in Abschnitt 3.1.3 aufgeführt werden, weshalb man Elasticsearch zu den NoSQL-Datenbanken zählen kann. Mit diesem auf Apache Lucene basierenden Suchserver, können neben Volltextinhalten auch strukturierte Daten verwaltet und schnell ausgelesen werden. Des weiteren passt Elasticsearch auch sehr gut zu Logstash, da beide Anwendungen von der gleichen Firma entwickelt werden.

In Kombination mit der Visualisierungs-Software Kibana bilden sie die drei Grundkomponenten des sogenannten „Elastic-Stack“ (zuvor auch „ELK-Stack“ genannt). Dieser Technologie-Stack wird sehr häufig auch von großen Unternehmen wie Facebook, Netflix oder Microsoft eingesetzt, um Daten aufzunehmen und sie in Echtzeit zu durchsuchen, zu analysieren und zu visualisieren. Dementsprechend wird der Stack auch immer weiter gepflegt, verbessert und um Anwendungen erweitert, die neue Funktionen bieten. So kann man beispielsweise mit „Shield“ Sicherheit auf Unternehmensniveau[1] realisieren oder zur proaktiven Überwachung die Software „Watcher“ einsetzen, welche Benachrichtigungen verschickt sobald ein konfiguriertes Ereignis eingetreten ist.

Ein weiterer Vorteil, der sich aus der weiten Verbreitung ergibt ist, dass es eine große Community gibt, weshalb die unterschiedlichen Technologien sehr gut dokumentiert sind und man für die meisten Problemen schnell Lösungen finden kann.

4.4 Data-Publishing

Auf die Daten im Webcache sollen später verschiedene Frontendanwendungen zugreifen können. Aus diesem Grund ist es notwendig, möglichst standardisierte Schnittstellen zu implementieren. Nach momentanem Stand der Technik sollte dies demnach eine RESTful-API sein (vgl. Abschnitt 3.4.1).

Eine API an dieser Stelle soll außerdem die dahinterliegenden Datenspeicher kapseln, damit sie zum einen austauschbar sind ohne das die Anwendungen, die über die API auf die Daten zugreifen, dadurch geändert werden müssen. Des weiteren kann eine solche Zwischenschicht auch wesentlich zur Sicherheit betragen. So könnten man hier auch Autorisierung und Authentifizierung implementieren. In den meisten Anwendungsfällen innerhalb der LUBW wäre es auch durchaus denkbar, nur lesende, also sichere Zugriffe über eine eigene API zu realisieren.

Um die unterschiedlichen Anforderungen der verschiedenen Anwendungsfälle abdecken zu können, sollten ganz im Sinne einer Microservice-Architektur (vgl. Abschnitt 3.4.2) viele kompakte Services implementiert werden. Im Idealfall pro abgetrenntem Funktionsbereich ein Service. Zum Beispiel ein Service, um über eine ID Stammdaten zu einer Station abrufen zu können oder ein Service der als Übergabeparameter eine geographische Lage erwartet und die ID's aller Station in einem bestimmten Umkreis zurückliefert. Des weiteren Services für Messdaten, die immer den aktuellsten Wert oder alle Werte in einem bestimmten Intervall liefern.

[1] schützt den gesamten Elastic Stack mit verschlüsselter Kommunikation, Authentifizierung, rollenbasierter Zugriffskontrolle und Auditierung

Die Schnittstelle sollte später nicht so implementiert werden, dass über eine bestimmte URI einfach nur die Daten für das entsprechende Frontend geliefert werden. Vielmehr sollten die unterschiedlichen Sevices durch die API miteinander vernetzt werden, wie es auch durch die HATEOAS-Eigenschaft gefordert wird.

Im Falle der Luft-Seiten beispielsweise wurden von der Fachseite bereits erste Mockups erstellt (siehe Abb. 2.6 auf Seite 9). Die beiden Seiten („Schadstoffe in Karten“ und „Schadstoffe in Tabellen“) stellen dabei zukünftig die Einstiegsseiten zu den Daten dar. Nach der HATEOAS-Eigenschaft sollten diese beiden Seiten ihre Daten über die gleiche Einstiegs-Ressource beziehen, welche dann die Daten in der jeweils passenden Repräsentation ausliefert. Für die Tabellen also beispielsweise in JSON und für die Karten in GeoJSON (vgl. Abschnitt 3.4.1).

Eine Einstiegs-Ressource sollte nach der HATEOAS-Eigenschaft jedoch keinesfalls alle Daten enthalten, die ein Nutzer vielleicht aufrufen könnte. Es genügt wenn Links zu allen relevanten Ressourcen enthalten sind. Will ein Nutzer mehr Informationen zu einer bestimmten Station, könnte die Daten für einen Infobox wie in Abbildung 2.6 zusehen, dann beispielsweise folgendermaßen besorgt werden. Zunächst wird die Ressource der entsprechenden Station aufgerufen. Darüber erhält man den Langnamen der Station. Den aktuellen Wert einer bestimmten Komponente und die Werte für den Wochenverlauf sollten wiederum eigene Ressourcen sein. Wobei die Links zum Abruf der entsprechenden Messdaten-Services in der Stations-Ressource hinterlegt sind.

4.5 Data-Visualizations

Mit allen in Abschnitt 3.3 beschriebenen Visualisierungs-Bibliotheken können generell die Anforderungen von Endanwendern an moderne Webdiagramme erfüllt werden (Anforderungen E 1 - E 5 in Abscnitt 2.2.3).

Wobei es bei den kommerziellen Lösungen Highcharts und amCharts bereits passende Vorlagen für alle benötigten Diagrammformen gibt, die - nur leicht angepasst - direkt verwendet werden können. Beide Lösungen bieten auch umfangreiche Konfigurationsmöglichkeiten und Schnittstellen, durch die man die Diagramme sehr individuell anpassen kann.

Die fertigen Diagramme sind bei allen Lösungen visuell sehr ansprechend, wobei die mit Highcharts erzeugten Diagramme eher seriös wirken und sich somit am besten für die Anwendungen einer Landesanstalt eignen. AmCharts-Diagramme hingegen wirken eher verspielt und sprechen eher eine jüngere Zielgruppe an.

Dank dem Visualisierungs-Framework FlexVis ist eine Entscheidung für oder gegen eine bestimmte Bibliothek jedoch nicht auf lange Zeit hin bindend. FlexVis bildet nämlich eine Abstraktionsschicht zur konkret eingesetzten Visualisierungslösung und kann mit beliebigen Bibliotheken zusammenarbeiten.

In der LUBW gibt es viele ähnliche Diagramme, die teilweise mit den unterschiedlichsten Techniken implementiert sind. Auch hier soll FlexVis Abhilfe schaffen. Damit können nämlich sowohl bestehender Quellcode von Visualisierungstypen (im wesentlichen HTML und Java-Script) als auch die unterschiedlichsten Datenquellen wiederverwendet werden (siehe Abschnitt 3.3.4). Es müssen somit nicht für jeden Fall neue Visualisierungen programmiert und neue Datenflüsse realisiert werden, was auch zu einem einheitlicheren Aussehen des LUBW-Webangebots beitragen wird.

Die Data-Sources einer Flexvis-Komponente können so erstellt werden, dass über Parameter die genaue Datenquelle zur Laufzeit geändert werden kann. Dadurch ist es möglich, auf einer Seite unterschiedliche Daten darzustellen. So muss man beispielsweise bei der Luft-Internetseite nicht für jede Station eine eigene Infoseite anlegen und pflegen. Es genügt eine Seite, auf der eine entsprechende FlexVis-Komponente platziert wird, welche dann alle Daten anzeigt die in einer Ressource über die gewünschte Station gespeichert sind.

Über eine andere Flexvis-Komponente kann so auch eine Seite realisiert werden, auf der alle Verlaufsgrafiken der Luft-Seiten dargestellt werden.

4.6 Betrieb

Heutzutage sollte eine Software-Architektur modular und generisch aufgebaut sein, um möglichst anpassbar zu sein, falls sich Anforderungen ändern oder eine Technologiewechsel nötig wird. Aus diesem Grund sollen die verschiedenen Komponenten, die in den vorherigen Abschnitten dieses Kapitels beschreiben werden, auch möglichst unabhängig von einander als einzelne Services implementiert werden.

Diese Microsevices sollen dann später in Docker-Containern (vgl. 3.4.3) auf einer hochverfügbaren Plattform betrieben werden. Wo genau diese Plattform betrieben wird - ob in der Cloud, im Rechenzentrum eines Providers oder bei der Landesoberbehörde IT Baden-Württemberg (BITBW) in einem Landesrechenzentrum - ist noch nicht endgültig entschieden. Aus diesem Grund sollte sie möglichst unabhängig von der darunterliegenden Infrastruktur aufgesetzt werden.

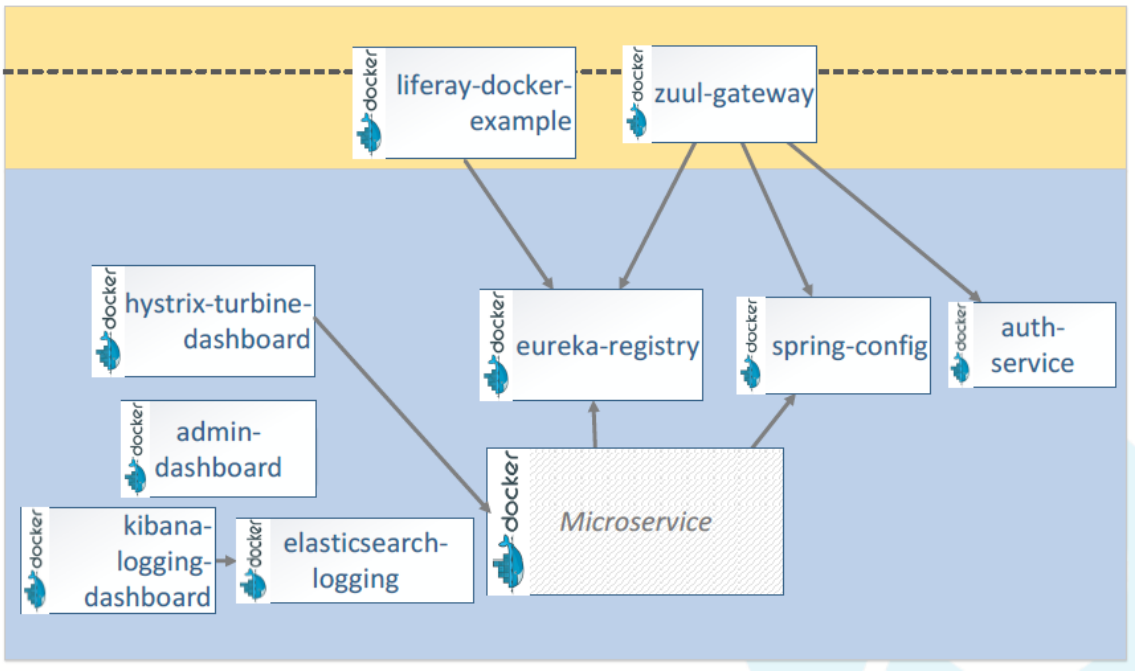

Abbildung 4.2: Aufbau einer Plattform zum Betrieb von Microservices [16]

Damit die einzelnen Services möglichst kompakt und simpel bleiben können, müssen Aufgaben wie beispielsweise Authentifizierung, Load-Balancing oder Circuit-Breaking[1] von der Plattform übernommen werden. Abbildung 4.2 zeigt den Aufbau der Plattform zum Betrieb von Microservices im Rahmen des Webcache der Kooperation Landesumweltportal.

Das „zuul-gateway“ bildet dabei die Eingangstür für alle Anfragen von Geräten und Webseiten an das Bachend. Zuul ermöglicht dabei dynamischen Routing, Überwachung und Sicherheit. [25]

Das „eureka-regestry“ fungiert als ein Register für die laufenden Microservices. Jeder Dienst registriert sich dort mit Information wo man ihn findet (Host, Port und Node-Name) und gegebenenfalls weiteren dienstspezifischen Metadaten. Benötigt eine Anwendung einen bestimmten Dienst, dann kann sie beim „eureka-regestry“ beispielsweise anfragen, ob es eine passenden Service gibt und wenn ja, wo dieser liegt. [23]

Das „hystrix-turbine-dashboard“ wird eingesetzt um den Gesundheitszustand aller Circuit Breaker zu überwachen. [34]

[1] Circuit Breaker werden benutzt um Fehler zu finden und kapseln die Logik um das wiederholte Auftreten eines Fehler zu verhindern(z.B. bei der Wartung oder temporären Ausfällen von externen Systemen). [40]

5 Prototypische Realisierung von Beispielszenarien

In diesem Kapitel wird die prototypische Umsetzung von zwei Beispielszenarien beschrieben.

In beiden Szenarien werden die Daten der Lärmmessstation bei Achern visualisiert. Bahnlärm hat sich in diesem Fall für eine prototypische Umsetzung angeboten, da diese Lärmmessstationen erst kurz vor Beginn dieser Bachelorarbeit an den Gleisen installiert wurden. Deshalb konnte beim Aufbau einer Infrastruktur zur Verarbeitung der Daten mit vergleichsweise geringem Aufwand ein Testsystem aufgesetzt werden. Auf diesem Testsystem liegen die Daten genau auf die gleiche Weise vor wie auf der produktiven Messstation. Zum einen die Rohdaten der Mikrofone in XML-Dateien. Zum anderen werden auf dem Testsystem aber auch die geplanten Tasks ausgeführt, die die Daten in regelmäßigen Zeitabständen in eine PostgreSQL-Datenbank überführen.

Als Ingestion-Tool zum Auslesen der Datenquellen wurde Logstash und als Webcache Elasticsearch verwendet. Die Visualisierungen der Daten wurden mit Highcharts realisiert.

5.1 Stunden-Pegel

Bei diesem Szenario liegen die Daten in einer PostgreSQL-Datenbank auf dem Testsystem und werden in einem Diagramm visualisiert, welches sich an der momentanen Umsetzung orientiert (siehe Abb. 5.1).

Wie der Datenfluss für diesen Prototyp aussieht, veranschaulicht Abbildung 5.2.

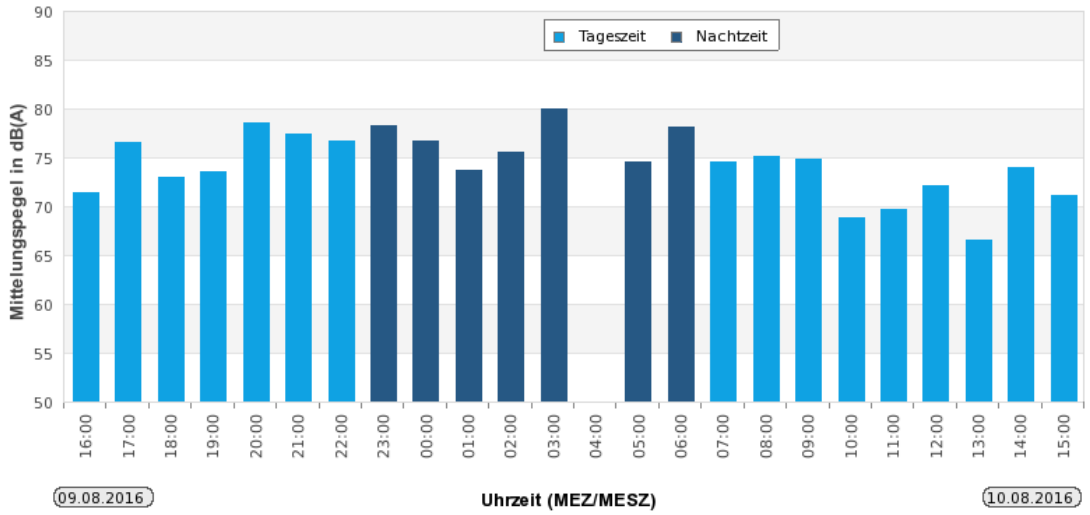

Abbildung 5.1: Momentane Visualisierung der Stundenpegel[1]

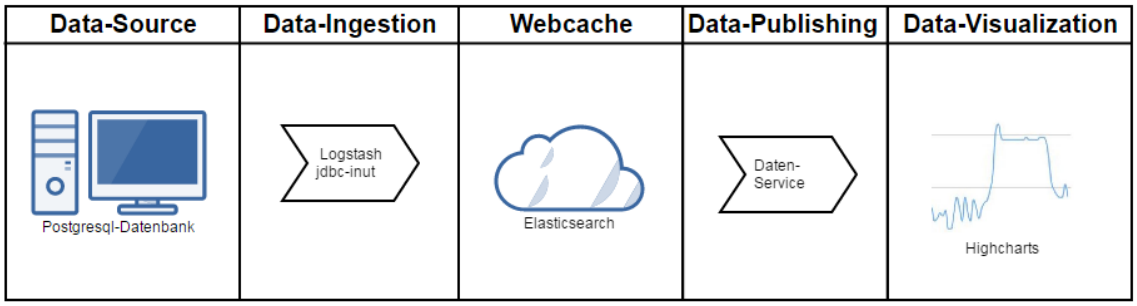

Abbildung 5.2: Datenfluss des Prototypen zur Anzeige von Stunde-Pegeln

5.1.1 Data-Ingestion

Listing 5.1 zeigt die Konfigurationsdatei für Logstash, die eingesetzt wurde um die Daten aus der Datenbank auszulesen und nach Elasticsearch zu transferieren.

```
input {
    jdbc {
        jdbc_driver_library => "/home/claus/Documents/postgresql
            -9.4.1208.jar"
        jdbc_driver_class => "org.postgresql.Driver"
        jdbc_connection_string => 'jdbc:postgresql://<ip-address>/
            mersy_pg'
        jdbc_user => "<user-name>"
        jdbc_password => "<password>"
        schedule => "15 * * * *"
```

[1] Bildquelle: http://www4.lubw.baden-wuerttemberg.de/servlet/is/258695/

```
9         jdbc_default_timezone => "Europe/Berlin"
10        statement => "select * from dralle_laerm_1h_temp where zeit > :
              sql_last_value order by zeit"
11    }
12 }
13 filter {
14    date {
15         match => ["zeit" , "ISO8601"]
16     }
17 }
18 output {
19         stdout { codec => rubydebug }
20       elasticsearch {
21           index => "laerm-1h"
22       }
23 }
```

Listing 5.1: Konfiguration von Logstash zum Auslesen einer Datenbank

Für den Zugriff auf die Datenbank wurde das jdbc-input-Plugin eingesetzt. Dazu wird zunächst der zur Java-Version und Datenbank passende jdbc-Driver benötigt. In der Konfigurationsdatei muss dann über den Parameter „jdbc_driver_library“ noch der Pfad angegeben werden wo die Bibliothek abgespeichert wurde und über den Parameter „jdbc_driver_class“ welche Klasse des Drivers geladen werden soll. [18]

Auf welche Datenbank zugegriffen werden soll, spezifiziert man über den „jdbc_connection_string“.

Ein weiteres Pflichtfeld dieses Plugins ist „jdbc_user“, über den man angibt, als welcher Benutzer man auf die Datenbank zugreift. Ist der Zugriff auf die Datenbank passwortgeschützt, kann man dieses über „jdbc_password“ eingeben.

Wann eine Abfrage auf die Datenbank durchgeführt werden soll, kann man über den Parameter „schedule“ konfigurieren. Wird hier nichts angegeben, wird die Abfrage nur einmal ausgeführt. Das Scheduling basiert auf dem Rufus-Scheduler[1], welcher Angaben in der Cron-Syntax entgegen nimmt. Damit können über fünf Felder sehr komplexe Ablaufplanungen realisiert werden. Über die ersten drei Felder kann man so zum Beispiel konfigurieren, zu welcher Minute, Stunde und an welchem Tag eine Abfrage durchgeführt werden soll. Setzt man für ein Feld „*“ ein, so ist es beliebig und trifft somit immer zu. In diesem Fall wird immer stündlich bei Minute 15 eine Abfrage durchgeführt. Dies wurde so gewählt um sicherzustellen, dass der Stunden-Mittelungspegel der zuletzt vergangen Stunde sich schon in der Datenbank befindet.

[1] https://github.com/jmettraux/rufus-scheduler

Der Parameter „jdbc_default_timezone“ ist optional, musste hier aber genutzt werden, da bei den Zeitstempeln in der Datenbank keine Zeitzone angeben ist.

Zuletzt wird hier noch in „statement“ die eigentliche Anfrage hinterlegt. Dabei wird der bei Logstash vordefinierte Parameter „sql_last_value“ verwendet, worüber man auf Werte der letzten Abfrage zugreifen kann. Damit wird gewährleistet, dass immer nur Datenbankeinträge ausgelesen werden, die in der Zwischenzeit hinzugekommen sind. In diesem konkreten Fall enthält die Variable den Zeitstempel der letzten Abfrage. Bei der ersten Abfrage auf eine Datenbank ist sie gesetzt auf den 1. Januar 1970 um 0:00 Uhr.

Da hier Daten direkt aus einer Datenbank ausgelesen werden, benötigt man keinen Filter um die gewünschten Werte auszulesen, dies kann alles über eine geeignete Abfrage realisiert werden. Logstash versieht jeden Datensatz jedoch automatisch mit einem Zeitstempel-Feld „@timestamp“, in welches der Zeitpunkt der Abfrage eingetragen wird. Nach diesem Feld kann man bei späteren Abfragen auf Elasticsearch einfach selektieren und sortieren. Um hier für jeden Datensatz den Wert zu hinterlegen, der auch in der Ur-Datenbank vermerkt ist, wird das date-filter-Plugin verwendet.

Der hier konfiguriete Output (ab Zeile 18) schreibt die Daten in eine lokale Elasticsearch unter den Index „laerm-1h“. Zum anderen werden die Daten für Debuggingzwecke auf dem Terminal ausgegeben.

5.1.2 Data-Publishing

Für dieses Anwendungsszenario wurde ein einfacher Service realisiert, durch den man die letzten 24 Stunden-Pegel in der Datenbank über einen HTTP-GET-Request an die URI „http://10.40.39.30:8090/laerm/DEBW4711/stunde“ eines lokalen Entwicklungsserver abfragen kann.

Implementiert wurde dieser Service mit dem Spring-Boot-Framework. Listing 5.2 zeigt den Quellcode.

Wie man erkennen kann, kapselt die Methode einfach nur einen HTTP-Post-Request auf die proprietäre HTTP-Schnittstelle von Elasticsearch. Dazu wird ein RestTemplate-Objekt verwendet, dem zum einen die URI zur Abfrage des Indexes „laerm-1h“ übergeben wird, außerdem die Abfrage im JSON-Format und die Information von welcher Klasse die Antwort sein wird (Zeile 26).

Über diese Abfrage (Zeile 23) werden alle Einträge unter dem Index angefordert jedoch abfallend sortiert nach „@timestamp“ und davon nur die ersten 24 Datensätze. D.h. also nur die neuesten 24 Werte.

Das Ergebnis dieser Abfrage wird dann direkt an den Client weitergeleitet, der die ursprüngliche Abfrage getätigt hat.

```
package com.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

import java.net.URI;

@RequestMapping("/laerm/DEBW4711/stunde")
@RestController
public class Laerm_1hController {
    @Autowired
    private RestTemplate rest;
    @CrossOrigin
    @RequestMapping(method = RequestMethod.GET, produces = {MediaType.
        APPLICATION_JSON_VALUE})
    public Object index() {
        URI uri = UriComponentsBuilder.fromUriString("http
            ://10.40.39.13:9200/laerm-1h/_search")
                .build()
                .toUri();

        String query =  "{\"sort\": { \"@timestamp\": { \"order\": \""+"
            incr"+ "\" }},"+"\"from\":\""+0+"\",\"size\":\""+24+"\"}";

        return rest.postForObject(uri, query, Object.class );
    }
}
```

Listing 5.2: Service zur Bereistelung der 24 letzten Stunden-Pegel [3] [35]

5.1.3 Data-Visualization

Zur Visualisierung wurde als Grundlage das Highchart Demo-Diagramm „Basic column“[2] verwendet.

[2] http://www.highcharts.com/demo/column-basic

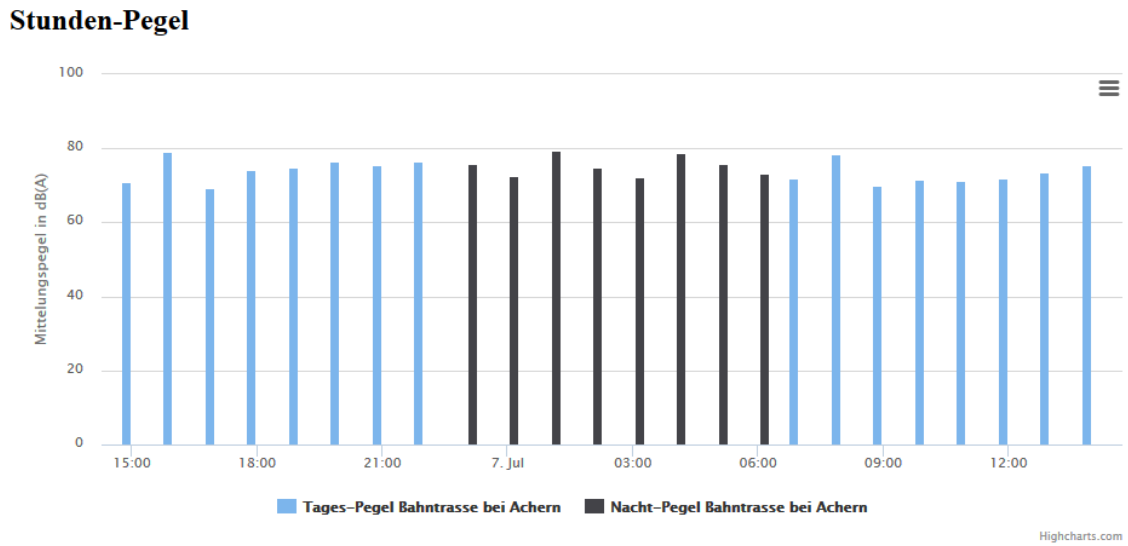

Abbildung 5.3: Prototypische Visualisierung der Stundenpegel

Abbildung 5.3 zeigt eine Screenshot eines solchen Diagramms. Wie man erkennen kann, ist es der momentanen Umsetzung sehr ähnlich. Auch hier werden die letzten 24 Stunden-Pegel gezeigt und Tages- und Nacht-Pegel unterschiedlich dargestellt.

```
var url = "http://10.40.39.30:8090/laerm/DEBW4711/stunde";

$.getJSON(url, function( data ){

var nachtDaten = [];
var tagDaten = [];

for (var elem in data.hits.hits){
var zeit = new Date(Date.parse(data.hits.hits[elem]._source.zeit))
  if (zeit.getHours() > 22 || zeit.getHours() <= 6){
    nachtDaten.push({
      x: zeit,
      y: parseFloat(data.hits.hits[elem]._source.laeq)
    })
  }else {
    tagDaten.push({
      x: zeit,
      y: parseFloat(data.hits.hits[elem]._source.laeq)
    })
  }
}
...
}
```

Listing 5.3: Visualisierung von Stunden-Pegeln (Auszug)

Listing 5.3 zeigt einen Auszug des Quellcodes (kompletter Quelltext siehe Anhang A) in dem die Daten mittels der jQuery-Funktion „getJSON“über die URL der API abgerufen werden. Außerdem zeigt der Auszug, wie die Datensätze ausgelesen werden und je nach Zeitstempel als Tages oder Nachtstempel klassifiziert werden. Die beiden resultierenden Objekt-Arrays „nachtDaten“ und „tagDaten“ werden dann später im Code jeweils als eigene Zeitreihe im Balkendiagramm dargestellt, wodurch von Highcharts automatisch eine farbliche Unterscheidung der beiden Klassen vorgenommen wird.

5.1.4 Integration in FlexVis

Dieser Visualisierungstype konnte in eine lokale Installation von FlexVis übertragen werden. Dazu musste der Visualisierungs-Code zunächst in die vorgegebene Struktur gebracht werden (vgl. Listing 3.6 auf Seite 31).

Damit das Diagramm über die GUI für Autoren editierbar ist, wurden drei Parameter im Code definiert. Über „Type“ kann der Diagrammtyp, über „color“ die Farbe der Balken für die Tages-Pegel und über „height“ die Höhe des Diagramms geändert werden. Für den kompletten Quellcode der fertigen Visualisierungs-Komponente siehe Anhang B.

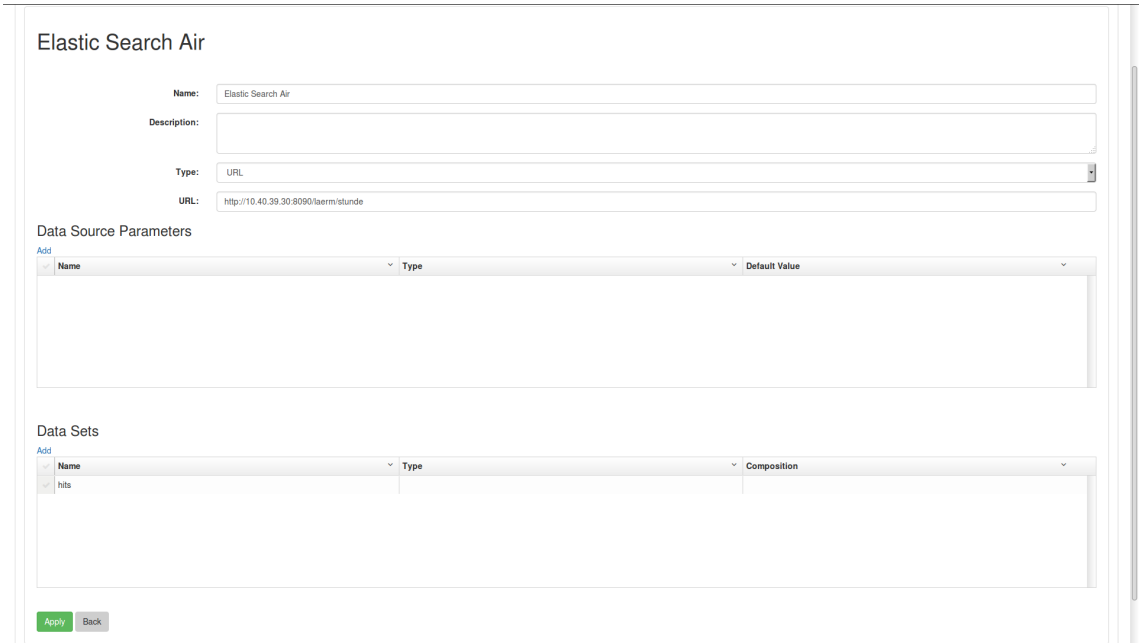

Abbildung 5.4: GUI zum anlegen eines Visualisierunstyps

Wie in Abschnitt 3.3.4 beschrieben, konnte diese Visualisierungs-Komponente anschließend über eine GUI eingepflegt werden (siehe Abb. 5.4). Hierzu muss man den Speicherort des Quellcodes angeben, des weiteren die Namen aller möglichen Dateneingänge sowie Name, Type und Standartwert aller möglichen Parameter.

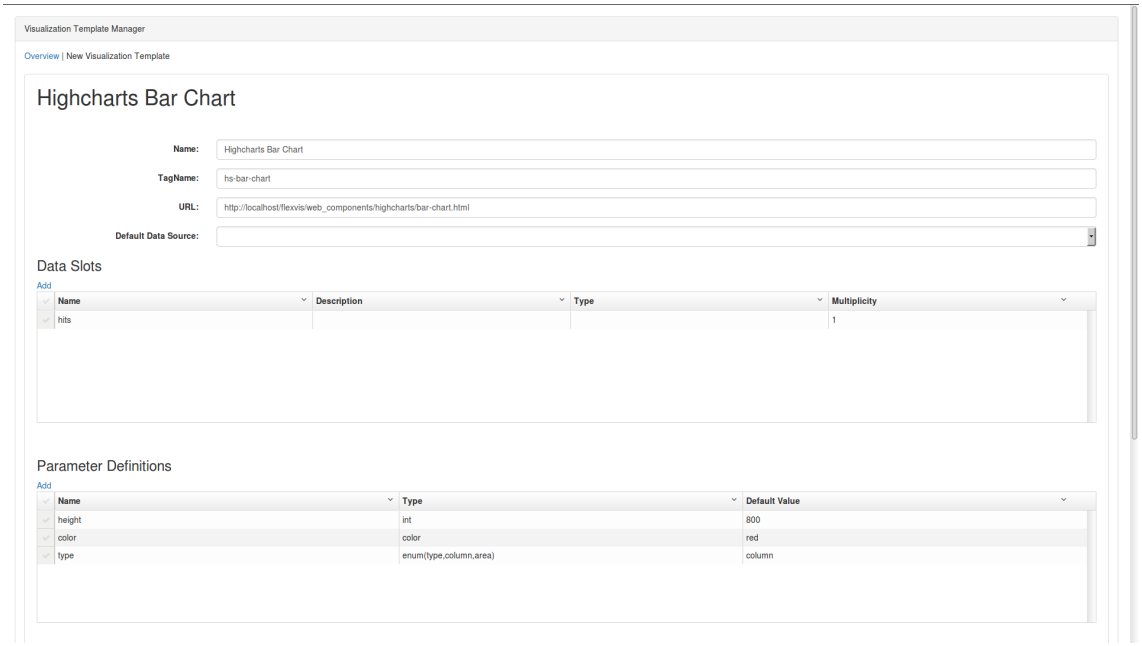

Abbildung 5.5: GUI zum anlegen einer Datenquelle

Abbildung 5.5 zeigt wie über die entsprechende GUI die Datenquelle in FlexVis eingepflegt wurde. In diesem Fall werden die Daten demnach über die URL „http://10.40.39.30:8090/laerm/stunde“ abgerufen und die empfangenen Daten bestehen nur aus einem Datensatz namens „hits“.

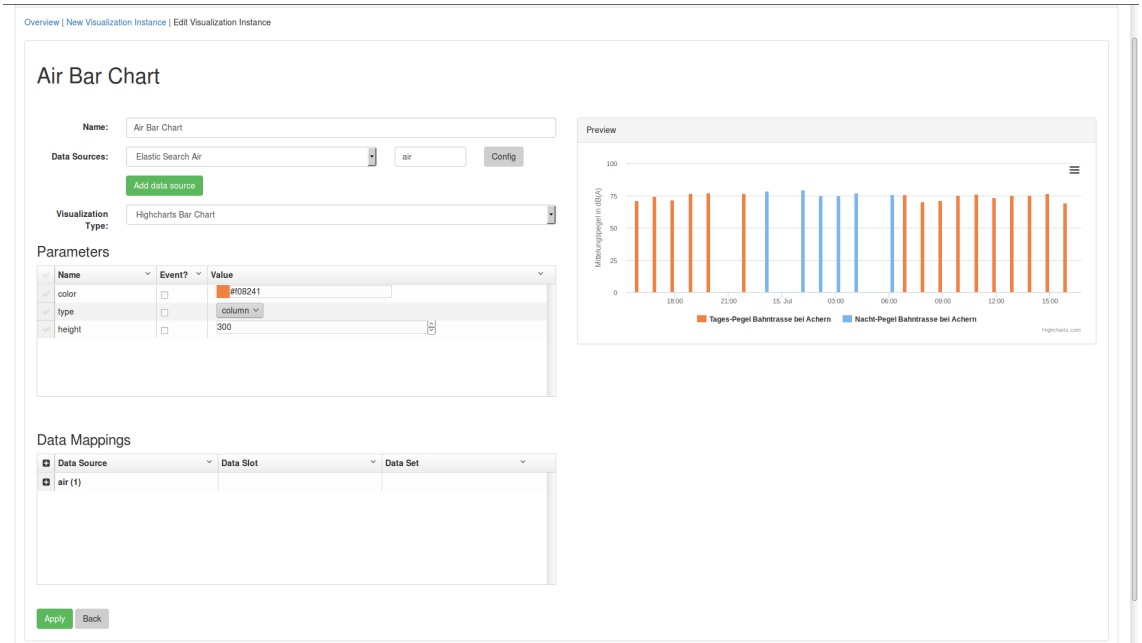

Abbildung 5.6: GUI zum anlegen einer Visualisierung

Wie die GUI aussieht, über die Redakteure Diagramme erstellen können zeigt Abbildung 5.6. Über Drop-Down-Menüs können sie unter den zuvor angelegte Visualisierungs-Typen und Datenquellen wählen. Des weiteren kann über Data-Mappings festgelegt werden, welche Datensätze aus der Datenquelle auf welchen Dateneingang der Visualisierung gelegt werden sollen. Zuletzt kann das Aussehen des Diagramms noch über die vorgesehenen Parameter angepasste werden.

5.2 Momentanpegel

Bei diesem Szenario werden die Daten aus XML-Dateien auf dem Testsystem gelesen. Anders als bei der momentanen Umsetzung werden hierbei die Pegel-Werte und die Informationen über den letzten Zug in nur einem Liniendiagramm dargestellt, anstatt in einem Linien- und einem Balkendiagramm (vgl. Abb. 5.7).

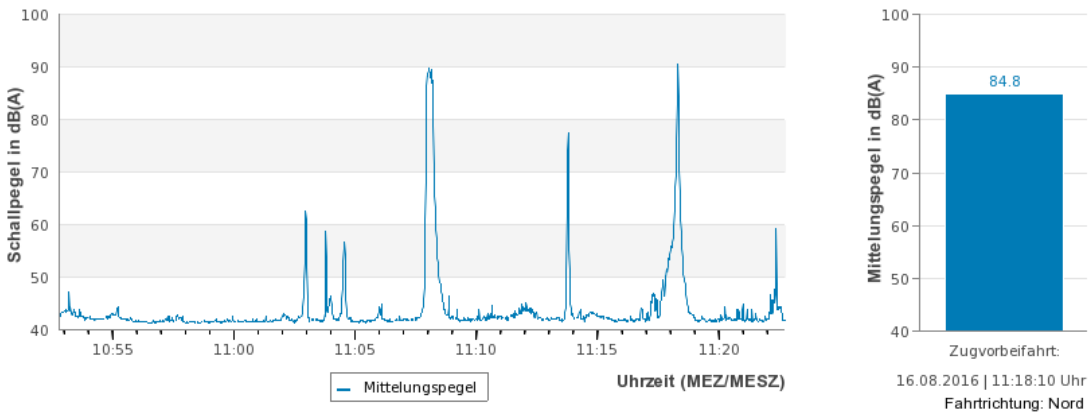

Abbildung 5.7: Momentane Visualisierung von Momentanpegeln[1]

Einen Überblick darüber, wie die fünf Architekturschichten bei diesem Prototyp konkret implementiert wurden, gibt Abbildung 5.8.

[1] Bildquelle: http://www4.lubw.baden-wuerttemberg.de/servlet/is/258692/

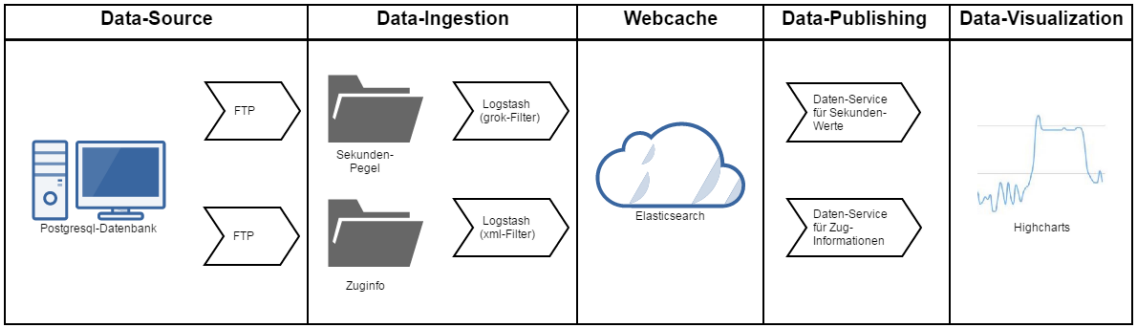

Abbildung 5.8: Datenfluss des Prototypen zur Anzeige von Momentanpegeln

5.2.1 Data-Ingestion

Bei diesem Prototypen stammen die Daten aus zwei unterschiedlichen XML-Dateien. Die benötigten Dateien werden dazu zunächst in festen Intervallen vom Testsystem auf einen lokal installierten FTP-Server geschoben. Auf diesen wiederum kann mit Logstash zur weiteren Verarbeitung zugegriffen werden.

In Abschnitt 5.2.1.1 wird die Logstash-Konfigurationsdatei beschrieben, die zum Auslesen der Sekunden-Werte benutzt wurde und in Abschnitt 5.2.1.2 die zum Auslesen der Zuginformationen.

5.2.1.1 Sekunden-Werte

Die Sekundenwerte befinden sich in XML-Dateien mit jeweils 60 Werten. Listing 5.4 zeigt einen Auszug aus einer solchen Datei. Wie man sieht befindet sich immer ein Datensatz in einer Zeile.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Values>
3    <Type>Second Values</Type>
4    <ProtocollVersion>1.34</ProtocollVersion>
5    <Location>2101_AkuMon</Location>
6    <StartDate>2016-07-15</StartDate>
7    <StartTime>07:27:46</StartTime>
8      <Data><Date>2016-07-15</Date><Time>07:27:46</Time><LAeq>45.9</LAeq><
           LAFmin>45.5</LAFmin><LAFmax>46.4</LAFmax><Status>0</Status></
           Data>
9      <!--...-->
10   <DataCount>60</DataCount>
11 </Values>
```

Listing 5.4: Datei mit Sekunden-Werten (Auszug)

Die entsprechenden Dateien liegen in einem lokalen Verzeichnis, dementsprechend wurde für den Input das „file“-Plugin gewählt, welches das angegebene Verzeichnis überwacht und registriert sobald eine Datei hinzukommt oder geändert wird. Durch das Setzen des Parameters „start_position“ aus „beginning“ erreicht man, dass beim ersten Lauf mit dieser Konfiguration alle Dateien durchsucht werden, die sich bereits im Ordner befinden.

```
1  input {
2      file {
3          path => "/Documents/Live-Daten/*"
4          start_position => "beginning"
5      }
6  }
7  filter {
8    grok {
9      match => {"message" => "<Data><Date>%{DATA:date}</Date><Time>%{DATA:
           time}</Time><LAeq>%{DATA:LAeq}</LAeq><LAFmin>%{DATA:LAFmin}</
           LAFmin><LAFmax>%{DATA:LAFmax}</LAFmax>"}
10   }
11   mutate {
12     add_field => ["Itimestamp", "%{date}T%{time}+02:00"]
13     remove_field => ["message"]
14     remove_field => ["date"]
15     remove_field => ["time"]
16     remove_field => ["path"]
17     remove_field => ["host"]
18   }
19   date{
20     match => ["Itimestamp" , "ISO8601"]
21   }
22 }
23 output {
24   if "_grokparsefailure" not in [tags] {
25     stdout {codec => rubydebug}
26     elasticsearch {
27       index => "laerm-live"
28     }
29   }
30 }
```

Listing 5.5: Konfiguration von Logstash zum Auslesen einer XML-Datei mit dem grok-Filter

Anschließend werden die neuen/bearbeiteten Dateien nach den gewünschten Informationen gefiltert. Dies erfolgt in diesem Fall mit dem „grok“-Filter (ab Zeile 8). Dabei handelt es sich um eine sehr vielseitigen Filter, über den man Eingaben mit eine regulärer Ausdruck abgleichen kann. Wird in einer Zeile dieser Ausdruck gefunden, können Ausschnitte davon

in Variablen gespeichert werden. Hier werden damit die Werte für Datum, Uhrzeit, sowie 1 s-Mittelungs-, Maximal- und Minimalpegel extrahiert.

Über den „mutate“-Filter (ab Zeile 11) wird anschließend aus Datum und Uhrzeit ein neues Feld „Itimestamp“ erzeugt und alle Felder entfernt, die nicht oder nicht mehr benötigt werden.

Mit dem „date“-Filter (ab Zeile 11) wird auch hier wieder der in der Datenquelle hinterlegte Zeitstempel („Itimestamp“) als offizieller Zeitstempel „@timestamp“ eingetragen.

Der hier konfiguriete Output (ab Zeile 23) schreibt die Daten in eine lokale Elasticsearch unter den Index „laerm-live“. Zum anderen werden die Daten für Debuggingzwecke auf dem Terminal ausgegeben. Das umschließende If-Statement stellt sicher, dass nur die Daten ausgegeben werden, die dem regulären Ausdruck aus dem „grok“-Filter entsprochen haben.

5.2.1.2 Zuginformationen

Auch hier liegen die Daten in einem lokalen Verzeichnis, weshalb auch hier als Input das „file“-Plugin gewählt wurde. Listing 5.6 zeigt einen Auszug aus einer XML-Datei, in der Informationen zu einem Zug übertragen werden.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Values>
3    <!-- ... -->
4    <Track1>
5      <TrainBeginDateTime>2016-07-14 17:01:48.680</TrainBeginDateTime>
6      <Duration>25.779</Duration>
7      <NoiseLevelSummary>
8        <LAFmin>64.89</LAFmin>
9        <LAFmax>87.43</LAFmax>
10       <LAeq>83.77</LAeq>
11       <!--...-->
12   </NoiseLevelSummary>
13   </Track1>
14   <!--...-->
```

Listing 5.6: Datei mit Zuginformationen (Auszug)

Die gewünschten Informationen können in diesem Fall nicht so einfach mit dem „grok“-Filter ausgelesen werden. Hier ist es geschickter den „XML“-Filter zu nutzen. Dazu müssen die Dateien jedoch als ganzes eingelesen werden und nicht zeilenweise, wie es standardmäßig geschieht.

Dass Logstash mehrere Zeilen zusammenfasst, kann über den Codec-Parameter „multiline“ eingestellt werden. In diesem Fall wird konfiguriert, dass wenn eine Zeile nicht mit „</Values>“ beginnt, dann gehört auch die nächste Zeile noch zu der Nachricht (ab Zeile 5).

```
1  input {
2      file {
3          path => "/home/claus/Documents/train-files/*"
4      start_position => "beginning"
5      codec => multiline {
6            pattern => "</Values>"
7            what => "next"
8            negate => "true"
9        }
10     }
11 }
12 filter {
13   xml{
14     source => "message"
15     store_xml => "false"
16     remove_namespaces => "true"
17     xpath => ["/Values/Track1/TrainBeginDateTime/text()","date"]
18     xpath => ["/Values/Track1/NoiseLevelSummary/LAeq/text()","wert1"]
19     xpath => ["/Values/Track1/Duration/text()","duration"]
20   }
21   mutate {
22     add_field => ["datum","%{date}+02:00"]
23     add_field => ["LAeq","%{wert1}"]
24     add_field => ["dauer","%{duration}"]
25     remove_field => ["duration"]
26     //...
27   }
28   date{
29     match => ["datum" , "ISO8601"]
30   }
31 }
32 output {
33   if "_dateparsefailure" not in [tags] {
34     stdout {codec => rubydebug}
35     elasticsearch {
36       index => "train-info"
37     }
38   }
39 }
```

Listing 5.7: Konfiguration von Logstash zum Auslesen einer XML-Datei mit dem XML-Filter

Das Feld „message“ enthält somit jeweils den gesamten Inhalt einer Zuginfo-XML-Datei. Mit dem „XML“-Filter-Plugin ist es dann möglich über „xpath“ Pfade entlang der XML-Baumstruktur abzulaufen. Damit werden hier die Felder „<TrainBeginDateTime>“, „<LAeq>“ und „<Duration>“ ausgelesen (ab Zeile 17).

Mit dem „mutate“-Filter (ab Zeile 21) werden anschließend wieder Felder angepasst und nicht benötigt Felder entfernt. Mit dem „date“-Filter (ab Zeile 32) wird auch hier wieder der in der Datenquelle hinterlegte Zeitstempel („datum“) als offizieller Zeitstempel „@timestamp“ eingetragen.

Der hier konfiguriete Output (ab Zeile 36) schreibt die Daten in eine lokale Elasticsearch unter den Index „train-info“. Zum anderen werden wieder die Daten für Debuggingzwecke auf dem Terminal ausgegeben. Das umschließende If-Statement stellt sicher, dass nur die Daten gespeichert werden, die eine gültigen Zeitstempel haben.

5.2.2 Data-Publishing

Für diesen Prototypen wurden zwei Messdaten-Services realisiert, durch die man die benötigten Daten über einen HTTP-GET-Request abfragen kann. Die Sekundenwerte erhält man über die URI „http://10.40.39.30:8090/laerm/DEBW4711/sekunde“ und die Zuginformationen über die URI „http://10.40.39.30:8090/laerm/DEBW4711/zug-info“.

Auch diese beiden Services wurden unter Nutzung des Spring-Boot-Frameworks umgesetzt. Die Implementierungen sind sehr ähnlich dem Dienst der in Abschnitt 5.1.2 beschrieben wurde. Eine Erweiterung hierbei ist jedoch, dass es hier möglich ist die Anfrage durch 4 Parameter zu konfigurieren.

```
1 public Object index(@RequestParam(value="startDate", defaultValue="now-2
      h") String startDate,
2             @RequestParam(value="endDate", defaultValue="now") String
                  endDate,
3             @RequestParam(value="order", defaultValue="incr") String
                  order,
4             @RequestParam(value="werteAnzahl", defaultValue="60") String
                   werteAnzahl) {
5 //...
6 }
```

Listing 5.8: Methoden-Definition eines parametrierbaren Services [3] [35]

Über die Parameter „startDate“ und „endDate“ kann ein zeitliches Intervall angegeben werden in dem die Werte liegen müssen. Hierzu müssen die Zeitpunkte gemäß der ISO-Norm 8601 mit einer Zeitzone angeben werden (siehe Abschnitt 3.1.5). Zu beachten gilt dabei, dass man das „+“ im Zeitstempel als „%2B“ kodieren muss, da dieses Zeichen

in einer URI nicht zulässig ist. Alternativ kann man auch Zeiten in Abhängigkeit zum aktuellen Zeitpunkt angeben (z.B. „now“ oder „now-2h“). Mit „order“ kann man die zeitliche Sortierung der Suchergebnisse ändern und mit „werteAnzahl“ wie viele Werte man erhalten möchte.

Für alle Parameter wurden Standardwerte festgelegt, die eingesetzt werden, wenn bei einer Anfrage der Parameter nicht definiert wurde. Wie die entsprechenden Methoden-Definition aussieht zeigt Listing 5.8. Dabei handelt es sich nur um einen Auszug, der komplette Quellcode beider Services befindet sich im Anhang (Anhang C und D).

5.2.3 Data-Visualization

Zur Visualisierung wurde als Grundlage das Highchart Demo-Diagramm „Spline updating each second“[1] verwendet.

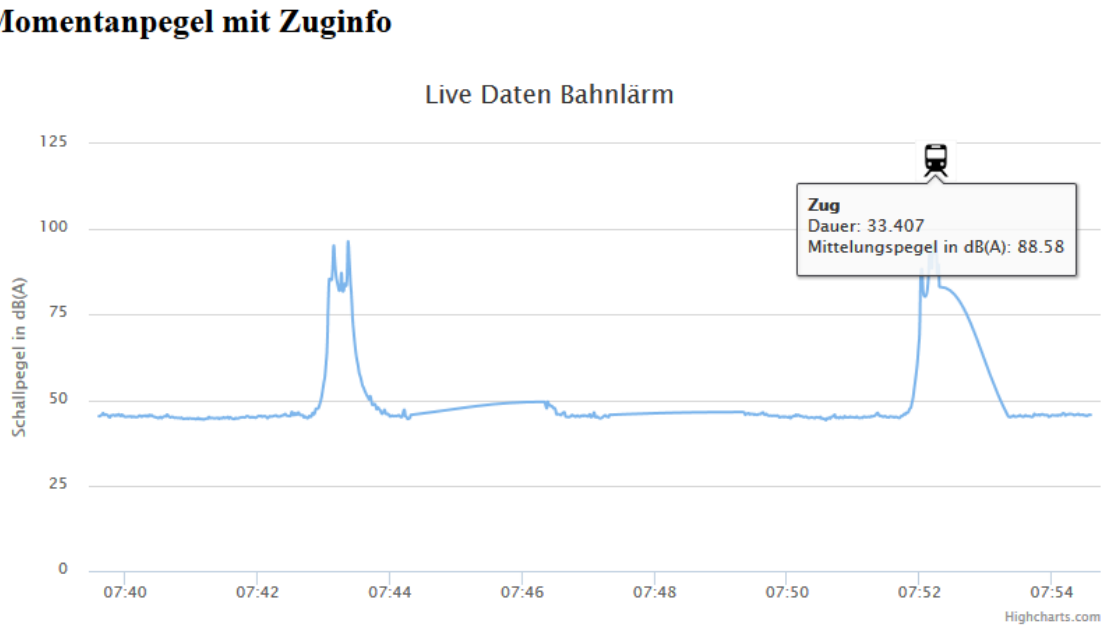

Abbildung 5.9: Prototypische Visualisierung von Momentanpegeln

Abbildung 5.9 zeigt eine Screenshot eines solchen Diagramms. Wie man erkenne kann sind hier die beiden momentanen Darstellungen in einem Diagramm zusammengefasst.

Listing 5.9 und 5.10 zeigen Auszüge des zur Visualisierung verwendeten Quellcodes (kompletter Quelltext siehe Anhang E). Listining 5.9 zeigt dabei die Funktion, welche die Datenreihe aktualisiert und Listing 5.10 die Funktion, welche die Informationen über den letzten Zug in das Diagramm einbringt. Beide werden jede Sekunde (1000 ms) von einer „setInterval“-Funktion aufgerufen.

[1] http://www.highcharts.com/demo/dynamic-update

```
1  setInterval(function () {
2
3    if (querying)return;
4
5    if (iterator <= 59){
6      time = Date.parse(dynDaten.hits.hits[iterator]._source.Itimestamp);
7      var x = time,
8        y = dynDaten.hits.hits[iterator]._source.LAeq;
9      series.addPoint([x, y], true, true);
10     iterator++;
11   }
12
13   if (iterator >= 60) {
14     querying = true;
15     start = time + 1000;
16     end = "now";
17     url = BaseUrl + "?startDate="+start+"&endDate="+end+"&werteAnzahl="
           +60+"&order=incr";
18     $.getJSON(url, function( data ){
19         dynDaten = data;
20         querying = false;
21         if (dynDaten.hits.hits[59] == null) return;
22         iterator = 0;
23
24     });
25   }
26
27   }, 1000);
```

Listing 5.9: Visualisierung von Momentanpegeln (Auszug)

Bei Funktion aus Listing 5.9 werden implizit drei Zustände des Diagramms vorausgesetzt: Der „Daten vorhanden“-, der „Keine Daten“- und der „Abfrage“-Zustand. Über die drei If-Statements wird abgefragt in welchem dieser Zustände sich das Diagramm befindet und die entsprechende Aktion ausgeführt.

Wenn keine Daten mehr vorhanden (Zeile 13) sind, wird zunächst die Variable „querying“ auf true gesetzt und anschließend die nächsten 60-Werte abgerufen (Data-Source liefert Messwerte in Paketen mit jeweils 60 Werten aus (vgl. Abschnitt 5.2.1.1)). Nach dem die Datenfrage erfolgreich abgeschlossen ist, wir „querying“ wieder auf false gesetzt. Wurde die Funktion in der Zwischenzeit nochmals aufgerufen, wurde über die Abfrage in Zeile 3 festgestellt, dass im Moment noch eine Abfrage stattfindet und die Funktion direkt wieder beendet. Sind noch Daten vorhanden, die noch nicht gezeichnet sind (Zeile 5), wird immer der nächste Datensatz gezeichnet, solange bis alle 60 Werte abgearbeitet sind.

Durch diese Art der Implementierung wird der Anschein erweckt, dass jede Sekunde ein neuer Wert geliefert wird, obwohl die ursprüngliche Datenquelle nur jede Minute neu Werte liefert.

```
setInterval(function(){
var endTrain = chart.series[0].data[zeitSpanneInSek-1].x;
var startTrain = chart.series[0].data[0].x;
var urlZug = BaseUrl + "/DEBW4711/zug-info?startDate="+ startTrain +"&
    endDate="+endTrain+"&werteAnzahl="+"1"+"&order=desc";

$.getJSON(urlZug, function( zugDaten ){
  if (zugDaten.hits.hits[0] == null){
    if (typeof chart.series[1] != 'undefined') chart.series[1].remove();
    return;
  }
  var datum = Date.parse(zugDaten.hits.hits[0]._source.datum);
  var dauer = parseFloat(zugDaten.hits.hits[0]._source.dauer);
  var markerPosition = datum + (dauer * 1000) / 2;

  if (typeof chart.series[1] != 'undefined') {
    if (markerPosition == chart.series[1].data[0].x) return;
    if (markerPosition >= endTrain) return;
    chart.series[1].remove();
  }
  if (markerPosition >= endTrain) return;

  chart.addSeries({
    name: "Zug <br/> Dauer: " + dauer + "<br/> Mittelungspegel in dB(A):
        " +
    zugDaten.hits.hits[0]._source.LAeq,
    data: [{
      x: markerPosition,
      y: 120,
      marker: {
        symbol: 'url(http://lazuli.voyage/websozai/gmapIcon/g_metro30.
            png)'
      }
    }]
  });
});
},1000)
```

Listing 5.10: Visualisierung von Momentanpegeln (Auszug)

Zur Anzeige der Zug-Informationen werden in der Funktion aus Listing 5.10 zunächst die Daten zum letzten Zug abgefragt, der innerhalb des momentan gezeichneten Zeitintervalls liegt.

Sollte es keine entsprechenden Daten geben, werden bereits gezeichnete Zugdaten, sofern vorhanden, gelöscht und die Funktion verlassen (Zeile 8-11).

Gibt die Abfrage Daten für einen Zug zurück wird daraus der Zeitpunkt berechnet, an dem der Zug-Marker im Diagramm angezeigt werden soll (Zeile 15).

Anschließend wird überprüft, ob die Zugdaten bereits eingezeichnet wurden. Wenn nicht und es ist bereits ein anderer Zug eingezeichnet, wird dieser zunächst gelöscht (Zeile 20), bevor der neue Zug ins Diagramm eingezeichnet wird (Zeile 25-35).

5.2.4 Integration in FlexVis

Bei FlexVis ist momentan noch nicht implementiert, dass man eine automatische Aktualisierung von Diagrammen über die GUI konfigurieren kann, weshalb diese Visualisierung für diesen Prototyp nicht in FlexVis integriert werden konnte.

Eine entsprechende Funktionalität ist aber nicht sehr aufwendig zu implementieren. Sie sollte deshalb bei einer nächsten Version hinzugefügt werden. Alternativ kann man eine Aktualisierung auch dadurch erreichen, dass man in bestimmten Intervallen über jQuery auf die FlexVis-Komponente zugreift und auf ihr die update-Funktion aufruft.

5.3 Reflexion der Prototypen

Bei den beiden Szenarien, die hier prototypisch umgesetzt wurden, wurden Daten aus zwei Datenquellen importiert (SQL-Datenbank und XML-Datei), die in Anforderung D 1 aufgeführt werden. Diese beiden Typen von Datenquellen kommen bei der LUBW am häufigsten vor.

Zur technischen Realisierung wurde als Webcache bei beiden Prototypen Elasticsearch gewählt. Diese ist eine mögliche Datenhaltungslösung für Messdaten, die für anderen Anwendungen bereits produktiv eingesetzt wird.

Die momentan im Internet veröffentlichten Darstellungsformen, konnten über die eingesetzten Visualisierungstechniken nachgebildet werden. Konkret wurde beim ersten Prototyp die Forderung nach Klassifizierung und entsprechender Kennzeichnung (Anforderung F 2) erfüllt. Außerdem wurde bei beiden Prototypen für die y-Achse eine fester Minimal- und Maximalwert definiert, wodurch die Forderung nach Vergleichbarkeit (Anforderung F 1) gewährleistet wird.

Neben diesen Grundanforderungen sind die Diagramme bereits von Haus aus responsiv und interaktiv (Anforderungen E 1 und E 4). So verändern sie je nach Bildschirmgröße die

Achsenskalierung und -beschriftung. Durch klicken auf die Legende, kann die entsprechende Zeitreihe ausgeblendet werden.

Beim zweiten Prototyp treffen minütlich neue Daten ein. Das entsprechende Diagramm aktualisiert sich automatisch auf visuell ansprechende Weise (Anforderung E 2).

Beim ersten Prototyp war bei den Zeitstempeln in der Datenbank keine Zeitzone angegeben. Dies führte bei der Umsetzung zunächst zu Fehlinterpretationen. Erst durch die explizite Anweisung in der Logstash-Konfigurationsdatei, für alle Zeitpunkte die Zeitzone „Europe/Berlin“ zu verwenden, konnte dieser Konflikt sauber behoben werden. Weiterhin problematisch ist die Ableitung des Zeitstempels beim zweiten Prototypen. Hier werden Datum und Tageszeit aus der XML-Datei gelesen und anschließend zu einem Zeitstempel nach der ISO-Norm 8601 (siehe 3.1.5) vereint. Dabei wird die Zeitzone statisch definiert als „+02:00“. Im Winter müsste man dies manuell auf „+01:00“ ändern.

Ein solche Vorgehensweise ist in einem produktiven System nicht akzeptabel. Hier bedarf es somit noch Besprechungsbedarf mit den Fachseiten, um eine möglichst fehlerunanfällige Lösung zu finden. Wie bereits in Abschnitt 3.1.5 beschrieben sollten Zeitstempel demnach immer mit einer Zeitzone versehen werden. Des weiteren könnte man Fehlinterpretationen vermeiden, wenn man immer eine festgelegt Zeitzone verwendet.

Eine weitere Punkt, der bei einem produktiven System so nicht übernommen werde darf, ist wie die URL-Parameter an Elasticsearch übergeben werden. Diese werden beim zweiten Prototyp nämlich einfach in Variablen ohne Überprüfung gespeichert und anschließend direkt ins Abfragestatement eingesetzt (vgl. Anhänge C und D). Eine solche Vorgehensweise ermöglicht eine SQL-Injektion und stellt somit eine große Sicherheitslücke da. In einem produktiven System müssen hier Prepared Statements eingesetzt und die Parameter validiert werden.

6 Fazit und Ausblick

Das primäre Ziel dieser Arbeit war es, eine Konzeption, für eine IT-Infrastruktur zur automatischen Bereitstellung und Darstellung von Umweltmessdaten im Internet, zu erstellen. Zur Validierung wurde diese Architektur an zwei Beispielszenarien prototypisch umgesetzt.

Dabei zeigte sich, dass die Architektur geeignet ist, um Daten aus verschiedenen Datenquellen in unterschiedliche Zielsysteme zu transferieren (Anforderungen D 1 und D 2).

Datenaggregation und -plausibilisierung (Anforderungen D 3 und D 4) wurden bei der Umsetzung nicht beachtet, da diese Punkte weitere Daten und/oder komplexe Fachlogik erfordern, weshalb eine genaue Betrachtung im zeitlichen Rahmen dieser Bachelorarbeit nicht möglich war. Da die Infrastruktur jedoch modular in mehreren Schichten aufgebaut ist, kann sie ohne größere Probleme nachträglich um entsprechende Module erweitert werden.

Durch den Einsatz von FlexVis können Redakteure später Diagramme über ein GUI erzeugen, Datenquellen zuordnen und editieren, sowie in beliebige Portale oder CMS integrieren (Anforderung D 5).

Liegen die Daten einmal im Webcache hochverfügbar vor, kann mit verschiedene Diensten eine Restful-API implementiert werden, über welche die verschiedenen Endanwendungen auf die Daten zugreifen können (Anforderung D 6).

Es wurden verschiedene Methoden diskutiert, wie ein Abgleich der Daten im Webcache mit den Daten der ursprünglichen Quellen realisiert werden könnte (Anforderung D 7). Vor einer konkreten Implementierung entsprechender Datenflüsse und Schnittstellen, muss jedoch erst eine Entscheidung für eine Methode gefällt werden.

Die in der Analyse der Webangebote identifizierten Diagrammtypen können durch den Einsatz moderner Visualisierungsbibliotheken nachgebildet werden (Anforderungen F 1 - F 3), wobei sie „out of the box“ responsiv und interaktiv sind (Anforderungen E 1 und E 4). Auch ein Aktualisieren der Diagramme ist über entsprechende Schnittstellen ohne weiteres möglich (Anforderung E 2).

Ein Diagramm, bei dem es möglich ist, in einen bestimmten Zeitbereich hineinzuzommen oder eine beliebiges Zeitintervall über eine Slider auszuwählen (Anforderung E 3), wurde in den beiden Beispielszenarien nicht beschrieben. Es gibt jedoch bei den betrachteten Visualisierungsbibliotheken Vorlagen, die genau diesen Anwendungsfall abdecken.

Es ist ohne weiteres möglich mehrere Datenreihen in einem Diagramm darzustellen. Um Anforderung E 5 zu erfüllen reicht dies jedoch nicht. Hierzu muss es möglich sein, dass ein Endnutzer die Datenreihen, die angezeigt werden, frei wählen kann. Dazu bedarf es einer eigenen Komponente, über die man eine Auswahl vornehmen kann. Diese müsste separat entwickelt werden.

Momentan ist die Umsetzung eines neuen Webauftritts für Luft bis Frühjahr 2017 angesetzt. Da die benötigte Infrastruktur sehr flexibel aufgebaut wird, wird eine Umstellung weitere Seiten mit Messdaten (z.B. die Lärm-Seiten) schneller und einfacher durchführbar sein.

Literatur

[1] *Acquiring Big Data Using Apache Flume*. URL: http://www.drdobbs.com/database/acquiring-big-data-using-apache-flume/240155029 (besucht am 16. 08. 2016).

[2] *Basic line | Highcharts*. URL: http://www.highcharts.com/demo/line-basic (besucht am 29. 07. 2016).

[3] *Building a RESTful Web Service*. URL: http://spring.io/guides/gs/rest-service/ (besucht am 12. 08. 2016).

[4] *C3.js | D3-based reusable chart library*. 1.05.2016. URL: http://c3js.org/gettingstarted.html (besucht am 29. 07. 2016).

[5] *Codec plugins | Reference [2.3] | Elastic*. 28.07.2016. URL: https://www.elastic.co/guide/en/logstash/current/codec-plugins.html (besucht am 28. 07. 2016).

[6] *Datenbanken / BASE | Datenbanken Online Lexikon*. URL: http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.BASE (besucht am 25. 07. 2016).

[7] *Datenbanken / CAP-Theorem | Datenbanken Online Lexikon*. URL: http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.CAP (besucht am 26. 07. 2016).

[8] *Datenbanken / MVCC - Multiversion Concurrency Control | Datenbanken Online Lexikon*. URL: http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.MVCC (besucht am 25. 07. 2016).

[9] *Definition » Internet der Dinge « | Gabler Wirtschaftslexikon*. URL: http://wirtschaftslexikon.gabler.de/Definition/internet-der-dinge.html (besucht am 24. 08. 2016).

[10] Michael Dienert. *Datums- und Zeitformate*. URL: http://dt.wara.de/pdf/sae/datenbanken/datumsformate/date.pdf (besucht am 21. 07. 2016).

[11] Stefan Edlich. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. 2., aktual. und erw. Aufl. München: Hanser, 2011. ISBN: 3446428550.

[12] Eric Braun. „A Highly Customizable and Generic Web Framework for Data Visualization“. Magisterarb. Karlsruhe Institute of Technology, 2015.

[13] *Flume or Kafka for Real-Time Event Processing*. URL: https://www.linkedin.com/pulse/flume-kafka-real-time-event-processing-lan-jiang (besucht am 27. 07. 2016).

[14] Martin Fowler und James Lewis. „Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr?“ In: (). URL: http://www.sigs-datacom.de/uploads/tx_mwjournals/pdf/fowler_lewis_OTS_Architekturen_15.pdf (besucht am 01. 08. 2016).

[15] *GeoAS - Das GeoInformationssystem (AGIS GmbH, Frankfurt am Main) - FME (Feature Manipulation Engine) - Die Datendrehscheibe*. URL: http://www.geoas.de/pages/de/software/fme.php (besucht am 27. 07. 2016).

[16] xdot GmbH. *LUBW Luftauftritt; WebCache, LUPO Rahmen-Architektur*. Internes Dokument. Telko am 28.07.2016.

[17] *Highcharts JS*. URL: https://shop.highsoft.com/highcharts (besucht am 29. 07. 2016).

[18] *jdbc | Reference [2.3] | Elastic*. 11.08.2016. URL: https://www.elastic.co/guide/en/logstash/current/plugins-inputs-jdbc.html (besucht am 11. 08. 2016).

[19] Markus Kramer. „NoSQL-Datenbanken“. In: (). URL: http://www.markus-kramer.de/blog/files/NoSQL-Datenbanken.pdf (besucht am 21. 07. 2016).

[20] Joe Kuan. *Learning Highcharts*. Community experience distilled. Birmingham: Packt Pub, 2012. ISBN: 9781849519083.

[21] *Logstash Introduction | Reference [2.3] | Elastic*. 28.07.2016. URL: https://www.elastic.co/guide/en/logstash/current/introduction.html (besucht am 28. 07. 2016).

[22] Michael Dazer. „RESTful APIs - Eine Übersicht“. In: (). URL: http://snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/restful-apis_dazer.pdf (besucht am 01. 08. 2016).

[23] *Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka*. URL: https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka (besucht am 25. 08. 2016).

[24] Dominik Nadberezny. *Aktualdaten Karten*. Internes Dokument.

[25] *Netflix/zuul*. URL: https://github.com/Netflix/zuul/wiki (besucht am 25. 08. 2016).

[26] *Online Store - amCharts*. URL: https://www.amcharts.com/online-store/ (besucht am 18. 08. 2016).

[27] Roy T. Fielding. *REST APIs must be hypertext-driven*. 1.08.2016. URL: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (besucht am 02. 08. 2016).

[28] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. Irvine: UNIVERSITY OF CALIFORNIA, 15.03.2002. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (besucht am 02. 08. 2016).

[29] Hermann Sauer. *Relationale Datenbanken - Theorie und Praxis*. München: Addison-Wesley Verlag, 2002. ISBN: 3-8373-2060-7.

[30] Edwin Schicker. *Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL*. 4., überarb. Aufl. Informatik et Praxis. Wiesbaden: Springer Vieweg, 2014. ISBN: 3834821853.

[31] Jochen Schnelle. *NoSQL – Jenseits der relationalen Datenbanken - Pro-Linux*. 19.08.2010. URL: http://www.pro-linux.de/artikel/2/1455/nosql-jenseits-der-relationalen-datenbanken.html (besucht am 17. 08. 2016).

[32] *Setting Up an Advanced Logstash Pipeline | Reference [2.3] | Elastic*. 15.08.2016. URL: https://www.elastic.co/guide/en/logstash/current/advanced-pipeline.html (besucht am 15. 08. 2016).

[33] Vaibhav Singhal. *20 best JavaScript charting libraries*. URL: http://thenextweb.com/dd/2015/06/12/20-best-javascript-chart-libraries/#gref (besucht am 29. 07. 2016).

[34] *Spring Cloud Netflix*. 25.08.2016. URL: http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html (besucht am 25. 08. 2016).

[35] *Spring REST Client with RestTemplate: Consume RESTful Web Service Example for XML and JSON*. URL: http://www.concretepage.com/spring/spring-mvc/spring-rest-client-resttemplate-consume-restful-web-service-example-xml-json#postforobject (besucht am 12. 08. 2016).

[36] *Time Series DBMS - DB-Engines Enzyklopädie*. URL: http://db-engines.com/de/article/Time+Series+DBMS (besucht am 26. 07. 2016).

[37] *Tutorial: Getting Started with FME Desktop - FME Knowledge Center*. URL: https://knowledge.safe.com/articles/1012/getting-started-with-fme-desktop-translate-data-be.html (besucht am 16. 08. 2016).

[38] *What is data ingestion? - Definition from WhatIs.com*. URL: http://whatis.techtarget.com/definition/data-ingestion (besucht am 27. 07. 2016).

[39] *What is Docker?* URL: https://www.docker.com/what-docker (besucht am 03. 08. 2016).

[40] Wikipedia, Hrsg. *Circuit breaker design pattern - Wikipedia, the free encyclopedia*. 1.08.2016. URL: https://en.wikipedia.org/w/index.php?oldid=725366434 (besucht am 16. 08. 2016).

[41] Wikipedia, Hrsg. *Docker (Software)*. 2.08.2016. URL: https://de.wikipedia.org/w/index.php?oldid=156383198 (besucht am 03. 08. 2016).

[42] Wikipedia, Hrsg. *HTML5*. 6.08.2016. URL: https://de.wikipedia.org/w/index.php?oldid=156284083 (besucht am 17. 08. 2016).

[43] Wikipedia, Hrsg. *ISO 8601*. 17.07.2016. URL: https://de.wikipedia.org/w/index.php?oldid=156218995 (besucht am 21. 07. 2016).

[44] Wikipedia, Hrsg. *Representational State Transfer*. 1.08.2016. URL: https://de.wikipedia.org/w/index.php?oldid=155740922 (besucht am 02. 08. 2016).

[45] Wikipedia, Hrsg. *Skalierbarkeit*. 12.08.2016. URL: https://de.wikipedia.org/w/index.php?oldid=156937490 (besucht am 15. 08. 2016).

[46] Wikipedia, Hrsg. *Time series database - Wikipedia, the free encyclopedia*. 14.07.2016. URL: https://en.wikipedia.org/w/index.php?oldid=729782581 (besucht am 26. 07. 2016).

Anhang

A. Kompletter JavaScript-Code zur Visualisierung von Stunden-Pegeln

B. Kompletter Quellcode der FlexVis-Visualisierungs-Komponente

C. Kompletter Java-Code des Services zur Bereitstellung von Sekundenwerten

D. Kompletter Java-Code des Services zur Bereitstellung von Zuginformationen

E. Kompletter JavaScript-Code zur Visualisierung von Momentanpegeln

A. Kompletter JavaScript-Code zur Visualisierung von Stunden-Pegeln

```
var url = "http://10.40.39.30:8090/laerm/DEBW4711/stunde";
var werteAnzahl = 24;
$(function () {

$.getJSON(url, function( data ){

var nachtDaten = [];
var tagDaten = [];

for (var elem in data.hits.hits){
var zeit = new Date(Date.parse(data.hits.hits[elem]._source.zeit))
  if (zeit.getHours() > 22 || zeit.getHours() <= 6){
    nachtDaten.push({
      x: zeit,
      y: parseFloat(data.hits.hits[elem]._source.laeq)
    })
  }else {
    tagDaten.push({
      x: zeit,
      y: parseFloat(data.hits.hits[elem]._source.laeq)
    })
  }
}
//Diagramm zeichnen
Highcharts.setOptions({
    global: {
      useUTC: false
    },
  });

$('#container').highcharts({
        chart: {
            type: 'column'
        },
        title: {
            text: null
        },
        xAxis: {
      type: 'datetime'
        },
        yAxis: {
            min: 0,
            title: {
```

```
                text: 'Mittelungspegel in dB(A)'
            }
        },
    legend:{
      enabled: true
    },
    tooltip: {
      headerFormat: '<span style="font-size:10px">{point.x}</span><table>',
      pointFormat: '<tr><td style="color:{series.color};padding:0">{series.name}: </td>' + '<td style="padding:0"><b>{point.y:.1f} dB(A)</b></td></tr>',
      footerFormat: '</table>',
      shared: true,
      useHTML: true
    },
    plotOptions: {
      column: {
        pointPadding: 0.2,
        borderWidth: 0,
      },
    },
    series: [
    {
      name: 'Tages-Pegel Bahntrasse bei Achern',
      data: tagDaten
    },
    {
    name: 'Nacht-Pegel Bahntrasse bei Achern',
    data: nachtDaten
    }
    ]
});
});
});
```

B. Kompletter Quellcode der FlexVis-Visualisierungs-Komponente

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>

```

```
<dom-module id="hs-bar-chart">
  <template>
    <h2>Stunden-Pegel</h2>
    </template>

  <script>

  Polymer({
            is: 'hs-bar-chart',
            visualization_start: function(data, params) {
        var children = Polymer.dom(this.root).childNodes;
        for(var i = 0; i < children.length; ++i) {
            Polymer.dom(this.root).removeChild(children[i]);
        }
            },
            visualization_update: function(data, params) {
              var children = Polymer.dom(this.root).childNodes;
                    for(var i = 0; i < children.length; ++i) {
                        Polymer.dom(this.root).removeChild(children[i]);
                    }
                this._draw(data, params);
            },
            ready: function () {

            },
            _draw: function(data, params) {
              var div = document.createElement("div");
              $(div).css("height", params.height + "px");
              Polymer.dom(this.root).appendChild(div);

              var nachtDaten = new Array;
    var tagDaten = new Array;

    for (var elem in data.hits.hits){
    var zeit = new Date(Date.parse(data.hits.hits[elem]._source.zeit))
      if (zeit.getHours() > 22 || zeit.getHours() <= 6){
        nachtDaten.push({
          x: zeit,
          y: parseFloat(data.hits.hits[elem]._source.laeq)
        })
      }else {
        tagDaten.push({
          x: zeit,
          y: parseFloat(data.hits.hits[elem]._source.laeq)
        })
      }
    }
```

```
52
53      Highcharts.setOptions({
54          global: {
55            useUTC: false
56          },
57
58        });
59
60      $(div).highcharts({
61        chart: {
62            type: params.type
63        },
64        title: {
65            text: null
66        },
67
68        xAxis: {
69            type: 'datetime'
70        },
71        yAxis: {
72            min: 0,
73            title: {
74          text: 'Mittelungspegel in dB(A)'
75            }
76        },
77          legend:{
78            enabled: true
79          },
80        tooltip: {
81            headerFormat: '<span style="font-size:10px">{point.x}</span><
                  table>',
82            pointFormat: '<tr><td style="color:{series.color};padding:0">{
                  series.name}: </td>' +
83          '<td style="padding:0"><b>{point.y:.1f} dB(A)</b></td></tr>',
84            footerFormat: '</table>',
85            shared: true,
86            useHTML: true
87        },
88        plotOptions: {
89            column: {
90          pointPadding: 0.2,
91          borderWidth: 0,
92            },
93          },
94        series: [
95          {
96            name: 'Tages-Pegel Bahntrasse bei Achern',
```

```
97            data: tagDaten,
98            color: params.color
99          },
100         {
101           name: 'Nacht-Pegel Bahntrasse bei Achern',
102           data: nachtDaten
103         }
104         ]
105         });
106     }
107   });
108   </script>
109 </dom-module>
```

C. Kompletter Java-Code des Services zur Bereitstellung von Sekundenwerten

```
1  package com.example;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.http.MediaType;
5  import org.springframework.web.bind.annotation.*;
6  import org.springframework.web.client.RestTemplate;
7  import org.springframework.web.util.UriComponentsBuilder;
8
9  import java.net.URI;
10 import java.util.Date;
11
12
13 @RequestMapping("/laerm/DEBW4711/sekunde")
14 @RestController
15 public class LaermController {
16
17   @Autowired
18   private RestTemplate rest;
19   @CrossOrigin
20   @RequestMapping(method = RequestMethod.GET, produces = {MediaType.
         APPLICATION_JSON_VALUE})
21   public Object index(@RequestParam(value="startDate", defaultValue="now
         -2h") String startDate,
22             @RequestParam(value="endDate", defaultValue="now") String
                   endDate,
23             @RequestParam(value="order", defaultValue="incr") String
                   order,
```

```
24              @RequestParam(value="werteAnzahl", defaultValue="60") String
                    werteAnzahl) {
25      URI uri = UriComponentsBuilder.fromUriString("http
            ://10.40.39.13:9200/laerm-live/_search")
26          .build()
27          .toUri();
28
29      String query = "{\"from\":\""+0+"\", \"size\":\""+werteAnzahl+"\","+
            "\"query\": {\"range\": { \"@timestamp\": {\"gte\":\""+
            startDate + "\",\"lte\":\""+ endDate + "\"}}},"+"\"sort\": { \"
            @timestamp\": { \"order\": \"" + order + "\" }}}";
30
31      return rest.postForObject(uri, query, Object.class );
32    }
33  }
```

D. Kompletter Java-Code des Services zur Bereitstellung von Zuginformationen

```
1  package com.example;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.http.MediaType;
5  import org.springframework.web.bind.annotation.*;
6  import org.springframework.web.client.RestTemplate;
7  import org.springframework.web.util.UriComponentsBuilder;
8
9  import java.net.URI;
10
11 @RequestMapping("/laerm/DEBW4711/zug-info")
12 @RestController
13 public class TrainInfoController {
14     @Autowired
15     private RestTemplate rest;
16     @CrossOrigin
17     @RequestMapping(method = RequestMethod.GET, produces = {MediaType.
           APPLICATION_JSON_VALUE})
18     public Object index(@RequestParam(value="startDate", defaultValue="
           now-1h") String startDate,
19                         @RequestParam(value="endDate", defaultValue="now
                               ") String endDate,
20                         @RequestParam(value="order", defaultValue="desc"
                               ) String order,
```

```
21                          @RequestParam(value="werteAnzahl", defaultValue=
                                "1") String werteAnzahl) {
22          URI uri = UriComponentsBuilder.fromUriString("http
                ://10.40.39.13:9200/train-info/_search")
23                  .build()
24                  .toUri();
25
26          String query = "{\"from\":\""+0+"\", \"size\":\""+werteAnzahl+"
                \","+"\"query\": {\"range\": { \"@timestamp\": {\"gte\":\""+
                 startDate + "\",\"lte\":\""+ endDate + "\"}}},"+"\"sort\":
                { \"@timestamp\": { \"order\": \"" + order + "\" }}}";
27
28          return rest.postForObject(uri, query, Object.class );
29      }
30  }
```

E. Kompletter JavaScript-Code zur Visualisierung von Momentanpegeln

```
1  var start = "now-2h";
2  var end = "now";
3  var pufferInSek = 60;
4  var zeitSpanneInSek = 600;
5  var iniWerteAnzahl = zeitSpanneInSek + pufferInSek;
6  var BaseUrl = "http://10.40.39.30:8090/laerm/DEBW4711/sekunde";
7
8  var url = BaseUrl + "?startDate="+start+"&endDate="+end+"&werteAnzahl="+
       iniWerteAnzahl+"&order=desc";
9
10 $(function () {
11
12 $.getJSON(url, function( data ){
13
14 var iterator = 60;
15 var querying = false;
16 var dynDaten = data;
17
18 //Ueberpruefen ob genuegend Daten vorhanden sind
19
20   if (dynDaten.hits.hits[iniWerteAnzahl-1]== null) {
21     alert("Nicht genuegend atuellen Daten vorhanden");
22     return;
23   }
24
```

```
var time = Date.parse(dynDaten.hits.hits[pufferInSek]._source.Itimestamp
    );

//Diagramm zeichnen

  Highcharts.setOptions({
    global: {
      useUTC: false
    }

  });

  $('#container').highcharts({
    chart: {
      type: 'spline',
      animation: Highcharts.svg, // don't animate in old IE
      marginRight: 10,
      events: {
        load: function () {

          var series = this.series[0];

          setInterval(function () {

            if (querying)return;

            if (iterator <= 59){
              time = Date.parse(dynDaten.hits.hits[iterator]._source.
                  Itimestamp);
              var x = time,
                y = dynDaten.hits.hits[iterator]._source.LAeq;
              series.addPoint([x, y], true, true);
              iterator++;
            }

            if (iterator >= 60) {
              querying = true;
              start = time + 1000;
              url = BaseUrl + "?startDate="+start+"&endDate="+end+"&
                  werteAnzahl="+60+"&order=incr";
              $.getJSON(url, function( data ){
                  dynDaten = data;
                  querying = false;
                  if (dynDaten.hits.hits[59] == null) return;
                  iterator = 0;

              });
```

```
69              }
70
71              }, 1000);
72          }
73        }
74      },
75      title: {
76        text: 'Live Daten Bahnl\u00e4rm'
77      },
78      xAxis: {
79        type: 'datetime',
80        /*dateTimeLabelFormats: {
81        second: '%d.%m.%Y - %H:%M:%S'
82        },*/
83        tickPixelInterval: 100
84      },
85      yAxis: {
86        title: {
87          text: 'Schallpegel in dB(A)'
88        },
89        max: 120,
90        min: 0,
91        plotLines: [{
92          value: 0,
93          width: 1,
94          color: '#808080'
95        }]
96      },
97      tooltip: {
98        formatter: function () {
99          if (this.series.name == 'Live-Daten Bahntrasse bei Achern')
100         return '<b>' + this.series.name + '</b><br/>' +
101           Highcharts.dateFormat('%Y-%m-%d %H:%M:%S', this.x) + '<br/>' +
102           Highcharts.numberFormat(this.y, 2);
103         else return '<b>' + this.series.name + '</b>';
104       }
105     },
106     legend: {
107       enabled: false
108     },
109     exporting: {
110       enabled: false
111     },
112     series: [{
113       id : 'messwerte',
114       name: 'Live-Daten Bahntrasse bei Achern',
115       data: (function () {
```

```
116
117         var data = [],
118           i;
119
120         for (i = iniWerteAnzahl-1; i >= pufferInSek; i--) {
121           data.push({
122             x: Date.parse(dynDaten.hits.hits[i]._source.Itimestamp),
123             y: dynDaten.hits.hits[i]._source.LAeq
124           });
125         }
126         return data;
127       }())
128     }]
129   });
130
131   var chart = $('#container').highcharts();
132
133   setInterval(function(){
134   var endTrain = chart.series[0].data[zeitSpanneInSek-1].x;
135   var startTrain = chart.series[0].data[0].x;
136
137   var urlZug = BaseUrl + "DEBW4711/zug-info?startDate="+ startTrain +"&
        endDate="+endTrain+"&werteAnzahl="+iniWerteAnzahl+"&order=desc";
138
139
140     $.getJSON(urlZug, function( zugDaten ){
141
142       if (zugDaten.hits.hits[0] == null){
143         if (typeof chart.series[1] != 'undefined') chart.series[1].
              remove();
144                 return;
145       }
146
147             var datum = Date.parse(zugDaten.hits.hits[0]._source.datum);
148             var dauer = parseFloat(zugDaten.hits.hits[0]._source.dauer);
149       var markerPosition = datum + (dauer * 1000) / 2;
150
151
152       if (typeof chart.series[1] != 'undefined') {
153         if (markerPosition == chart.series[1].data[0].x) return;
154         if (markerPosition >= endTrain) return;
155         chart.series[1].remove();
156       }
157
158       if (markerPosition >= endTrain) return;
159
160             chart.addSeries({
```

```
161                 name: "Zug <br/> Dauer: " + dauer + "<br/>
                        Mittelungspegel in dB(A): " +
162                 zugDaten.hits.hits[0]._source.LAeq,
163                 data: [{
164                     x: markerPosition,
165                     y: 120,
166                     marker: {
167                         symbol: 'url(http://lazuli.voyage/websozai/
                            gmapIcon/g_metro30.png)'
168                     }
169                 }]
170             });
171     });
172   },1000)
173 });
174 });
```