



**University of
Zurich^{UZH}**

Konzeption und Implementierung einer interaktiven Visualisierungsapplikation mit Datenbankanbindung

Bachelorarbeit im Fach Informatik

vorgelegt von:

Manuel Keller, Wil SG

13-795-125

**Angefertigt am
Institut für Informatik
der Universität Zürich
Prof. Dr. Lorenz M. Hilty**

Betreuer: Dr. Sinan Teske

Abgabezeitpunkt: 04.10.2016

Abstract

Die Empa Dübendorf arbeitet an einer Studie, welche die Wirtschaftlichkeit von „Power to Hydrogen / Hydrocarbon“ Technologien erforscht. Diese Technologie wandelt Elektrizität in Methangas um, welches im hochpreisigen Mobilitätsmarkt verkauft werden kann. Mit den Einkünften können die Deckungsbeiträge an die Infrastruktur erneuerbarer Energien bezahlt werden.

Im Rahmen dieser Bachelorarbeit wurde der Autor der Studie unterstützt. Es wurde ein Datenmodell erarbeitet, mit welchem die im Rahmen der Studie anfallenden Daten in einer Datenbank gespeichert werden können.

Darauf aufbauend wurde ein Visualisierungstool implementiert, mit welcher die Daten für die Anwendung in der Studie graphisch aufbereitet werden können. Die Umsetzung erfolgte als interaktive Webapplikation.

Danksagung

Ich möchte mich bei all jenen bedanken, welche mich in der Erarbeitung dieser Bachelorarbeit unterstützt haben.

Speziellen Dank gebührt Prof. Dr. Lorenz M. Hilty und Dr. Sinan Teske. Sie gaben mir die Möglichkeit, an diesem spannenden Projekt an der Empa mitarbeiten zu können. Ich will mich auch für die konstruktiven Treffen bedanken, deren Kritik und Lob einiges zur Qualität der erarbeiteten Ergebnisse beigetragen haben.

Ich möchte mich auch bei meinen Mitbewohnern Dimitri Kohler und Florian Fuchs bedanken, welche mich mit Programmiertipps unterstützt haben.

Inhaltsverzeichnis

ABSTRACT	III
DANKSAGUNG	V
KAPITEL 1 EINLEITUNG	1
1.1 ZIELSETZUNG	2
KAPITEL 2 DEFINITIONEN	3
2.1 SCRUM	3
2.2 NoSQL	4
2.3 R (PROGRAMMIERSPRACHE)	5
2.4 REAKTIVE PROGRAMMIERUNG	5
2.5 SHAPEFILE	6
KAPITEL 3 KONZEPTION	7
3.1 VORGEHENSWEISE	7
3.2 ANFORDERUNGEN AN DAS DATENMODELL	7
3.3 ANFORDERUNGEN AN DAS VISUALISIERUNGSTOOL	8
KAPITEL 4 EVALUIERUNG DATENBANK	10
4.1 RELATIONALE DATENBANK	10
4.2 NoSQL: GRAPHDATENBANK	11
4.3 AUSWERTUNG	12
KAPITEL 5 EVALUIERUNG VISUALISIERUNGSTOOL	13
5.1 BESTEHENDE VISUALISIERUNGSTOOLS	14
5.1.1 Tableau Desktop	14
5.1.2 Microsoft Power BI	14
5.1.3 Qlik QlikView	15
5.1.4 Eigenentwicklung	16
5.2 AUSWERTUNG	17
KAPITEL 6 UMSETZUNG	18
6.1 VORGEHENSWEISE	18

6.2 DATENMODELL.....	19
6.2.1 Beschreibung der Objekttypen	20
6.2.2 Aufbau Beispieldatenbank.....	21
6.3 PROTOTYP STROMERZEUGUNG.....	22
6.3 VISUALISIERUNGSTOOL.....	24
6.3.1 eingesetzte Technologien	26
6.4 IMPLEMENTATIONSDetails.....	27
6.4.1 Datenmodell Visualisierungstool.....	27
6.4.2 Data Overview Visualisierung.....	28
6.4.3 Daten laden	28
6.4.4 Filtern	28
6.4.5 Data Exploration.....	30
6.4.6 Dynamisches UI-Element.....	31
6.4.7: Visualisierungscode.....	31
6.4.8 Export	32
6.4.9 Gliederung des Codes.....	33
6.4.10 Code-Kommentare	34
KAPITEL 7 DISKUSSION DER RESULTATE	35
7.1 EIGNUNG DES DATENMODELLS.....	35
7.2 EIGNUNG DES VISUALISIERUNGSTOOLS	35
8.3 AUSBLICK	36
QUELLENVERZEICHNIS.....	38
ABBILDUNGSVERZEICHNIS.....	43
TABELLENVERZEICHNIS.....	45
PROGRAMMCODEVERZEICHNIS.....	46
ANHANG A SCREENSHOTS DER WEBAPPLIKATION	48
ANHANG B BENUTZERHANDBUCH VISUALISIERUNGSAPPLIKATION	52
B.1 DATENBANKMANAGEMENT.....	52
B.1.1 Zeitreihendaten in Datenbank einfügen	53
B.1.2 Geodaten / Shapefiles in Datenbank einfügen	55
B.1.3 Daten löschen.....	56
B.2 VISUALISIERUNGSTOOL	57
B.2.1 Tab Data Overview	57
B.2.2 Tab Data Explorer	59

<i>B.2.3 Tab Visualization</i>	<i>60</i>
<i>B.2.4 Tab Raw Data.....</i>	<i>63</i>
ANHANG C LISTE DER VERBESSERUNGEN	64
C.1: FUNKTIONALITÄTEN	64
C.2 REFACTORING	64
INHALT DER CD	66

Kapitel 1

Einleitung

In der Herbstsession 2016 entscheidet das Schweizer Parlament über die Energiestrategie 2050 (Bundesamt für Energie, 2016, S. 27). Ein wichtiger Punkt dieser Strategie ist der Verzicht auf einen Ersatz von Kernkraftwerken. Deren Stromproduktion soll durch erneuerbare Energien ersetzt werden, was das Elektrizitätsnetz der Schweiz vor grosse Herausforderungen stellt. Im Bericht „Versorgungssituation Schweiz 2016“ der Eidgenössischen Elektrizitätskommission ist denn auch zu lesen „dass die Anforderungen an den Systembetrieb aufgrund der sich verändernden Lastflüsse, insbesondere auch aufgrund der zunehmenden Produktionskapazität mit fluktuierender Einspeisung, in Zukunft tendenziell zunehmen.“ (ElCom, 2016, S. 64).

Besonders die Wasserkraft ist von einem starken Wandel betroffen. Flüsse führen im April bis September deutlich mehr Wasser als im Rest des Jahres, weswegen zu diesen Zeiten auch mehr Elektrizität aus Wasserkraft erzeugt wird. Durch die gleichzeitig zunehmende Einspeisung von Elektrizität aus Photovoltaik sinken die Strommarktpreise im Sommer aber immer stärker, sodass die Wasserkraftwerke trotz Subventionen keinen ausreichenden Deckungsbeitrag zu Erhaltung ihrer Infrastruktur erhalten (Geissmann, 2016). Nicht umsonst hat der Energiekonzern Alpiq im März 2016 medienwirksam angekündigt, einen Teil seiner Wasserkraftwerke zu verkaufen, da diese zu unrentabel seien (Tagesanzeiger, 2016).

An dieser Stelle setzt Dr. Sinan Teske an. Er arbeitet an einer Machbarkeitsstudie des Bundes zum Thema „Power to Hydrogen (Wasserstoff) / Hydrocarbon (Methan)“ Technologien in der Schweiz mit Fokus auf den Mobilitätsbereich. In dieser soll erforscht werden, ob und wann es sich lohnt, überschüssige erneuerbare Elektrizität in Wasserstoff und/oder Methan umzuwandeln. Interessant ist, dass die Produkte dieser Umwandlung im Mobilitätssektor verkauft werden können. Dort sind die Preise höher als auf dem Elektrizitätsmarkt, weshalb die Deckungskosten von erneuerbaren Energien eventuell gedeckt werden könnten. Als Zusatz würde dies die Abhängigkeiten von Gasimporten senken.

Dem Leser der Studie sollen die im Rahmen der Studie gesammelten Daten durch Visualisierungen veranschaulicht werden. In Abendnachrichten, Zeitungen, aber auch am Arbeitsplatz sind Visualisierungen häufig anzutreffen. In verschiedenen Formen helfen sie uns, die Welt um uns herum zu verstehen, indem sie abstrakte Daten und Zusammenhänge graphisch veranschaulichen. Im Gegensatz zu

textuellen Zusammenhängen können graphische parallel und damit deutlich schneller erfasst werden.

Im Rahmen meiner Bachelorarbeit habe ich Dr. Sinan Teske bei seiner Arbeit unterstützt, indem ich die Speicherung der Daten und ein Visualisierungstool konzeptioniert habe. Diese schriftliche Arbeit dokumentiert die Konzeptionierung und die getroffenen Evaluationsentscheidungen. Sie erklärt die Implementierung des Visualisierungstools und enthält zudem ein Benutzerhandbuch zur Applikation. Der Aufbau ist chronologisch und beginnt mit der Konzeption, in welcher die Anforderungen geklärt werden. Im Anschluss wurden Evaluationen zu Datenbank und Visualisierungssoftware durchgeführt und die schlussendlich gewählte Lösung näher erläutert.

1.1 Zielsetzung

Die in Dr. Sinan Teskes Studie gesammelten Daten sollen für Visualisierungszwecke verfügbar gemacht werden. Das Ziel dieser Bachelorarbeit besteht also darin, ein Datenmodell zur Speicherung von Daten im Energiesektor auszuarbeiten. Daten mit gemeinsamen Aspekten, wie Zeit oder Ort, sollen vergleichbar gespeichert werden. Das Datenmodell soll in einer Datenbank realisiert werden.

Diese konsolidiert gespeicherten Daten sollen für Datenanalysen und Visualisierungen zugänglich sein. Dazu soll eine Visualisierungsapplikation konzeptioniert und implementiert werden. Gute Performance und interaktive Bedienung sollen den Prozess beschleunigen und ansprechend gestalten. Die Resultate sollen in wissenschaftlichen Arbeiten benutzt werden.

Mit dem Visualisierungstool sollen durch Kombinieren und Vergleichen neue Erkenntnisse gewonnen werden können, welche nicht auf den ersten oder zweiten Blick ersichtlich sind. Die Daten sollen deswegen explorativ durchsuchbar sein. So können noch nicht berücksichtigte, eventuell relevante Daten in die Visualisierungen eingebunden werden.

Kapitel 2

Definitionen

2.1 SCRUM

SCRUM ist eine in der Software-Entwicklung weit verbreitete Vorgehensweise. „[SCRUM] ist weder ein Prozess noch eine Technik zur Erstellung von Produkten, sondern ist vielmehr als Rahmenwerk zu verstehen, innerhalb dessen verschiedene Prozesse und Techniken zum Einsatz gebracht werden können.“ (Schwaber & Sutherland, 2013, S. 3). Das wichtigste Element der SCRUM Vorgehensweise ist die kontinuierliche Kommunikation zwischen Entwicklungsteam und Auftraggeber. Indem neue Entwicklungen, Ideen und Funktionen in einem definierten Zeitabstand dem Auftraggeber präsentiert werden, können Differenzen zwischen gewünschtem und tatsächlich abgeliefertem Produkt frühzeitig erkannt und korrigiert werden. Durch den stetigen Austausch wird der Auftraggeber aktiv in den Entstehungsprozess eingebunden und kann mitentscheiden, wo Prioritäten gesetzt werden müssen.

Der Entwicklungsprozess teilt sich in Sprints auf: Jeder Sprint stellt eine zeitlich definierte Entwicklungs-Etappe dar, welche zu Beginn geplant und am Ende beurteilt wird. Die Planung erfolgt auf Basis eines „Product Backlog“ (Liste mit abzuarbeitenden Aufgaben), das alle Anforderungen der Anspruchsgruppen enthält. Basierend auf Aufwands-Schätzungen der Aufgaben entscheidet das Entwicklungsteam, wie viele der Backlog-Items im nächsten Sprint bearbeitet werden. Dies ergibt das „Sprint Backlog“, welches vom Entwicklungsteam abgearbeitet wird. Am Ende jedes Sprints erfolgt eine Beurteilung, anhand welcher der Entwicklungsprozess optimiert wird.

In der SCRUM Vorgehensweise wird an einem „Inkrement“ gearbeitet. Ein Inkrement ist das Resultat aller abgeschlossenen Sprints und stellt eine lauffähige Zwischenfassung des Endprodukts dar. Das Produkt, an dem entwickelt wird, ist also stets in einem verwendbaren Zustand. Dies bedeutet, dass es getestet und beurteilt werden kann. Am Ende jedes Sprints wird das Inkrement dem Auftraggeber übergeben.

SCRUM definiert drei Rollen (Schwaber & Sutherland, 2013):

- **Product Owner:** Auftraggeber oder Stellvertreter; gibt Ziele vor
- **Entwicklungsteam:** Implementierer; setzt Ziele um
- **SCRUM Master:** Überwachungsorgan; verantwortlich für die Einhaltung der SCRUM Regeln

2.2 NoSQL

NoSQL steht für „Not only SQL“ und nicht etwa „No SQL“. SQL wird also nicht abgelehnt. Es wird allerdings in Frage gestellt, ob klassische, relationale Datenbanken tatsächlich alle Speicherprobleme bestmöglich lösen können.

Vielfach greifen die als NoSQL bezeichneten Datenbanken alte Konzepte auf, welche durch neue Entwicklungen und Anforderungen wieder interessant geworden sind. Dass mittlerweile eine wachsende Zahl grosser Internetunternehmen solche Datenbanken benutzen, zeigt eindeutig die Relevanz der Thematik. So haben beispielsweise Google, Facebook und Amazon jeweils eine eigene NoSQL Datenbank implementiert, die ihre spezifischen Anforderungen abdecken (Garling, 2012). Die Firmen sehen die Vorteile in der besseren Bewirtschaftung grosser Datenmengen, die mit relationalen Datenbanken kaum mehr zu bewerkstelligen wäre. Momentan gibt es über 200 Datenbanksysteme, die der NoSQL Strömung angerechnet werden können (NoSQL Databases, 2016). Die wichtigsten Kategorien darin sind:

Kategorie	Merkmale
Dokumentenorientierte Datenbanken	<ul style="list-style-type: none">- Speichert unstrukturierte Textdaten in beliebiger Länge- zusammengehörige Daten werden zusammen gespeichert <p>(Gull, 2012, S. 20)</p>
Graphen-Datenbanken	<ul style="list-style-type: none">- Spezialisiert auf vernetzte Informationen- Erlaubt die Speicherung von Daten und deren Beziehungen untereinander <p>(Graphendatenbank, 2013)</p>
Key-Value-Datenbanken	<ul style="list-style-type: none">- Speichert Werte unter einem Schlüssel, durch die Einfachheit extrem skalierbar <p>(Gull, 2012, S. 20)</p>
Spaltenorientierte Datenbanken	<ul style="list-style-type: none">- Speichern Dateneintrag in einer Spalte, nicht in einer Zeile- Ähnliche Einträge werden als „Column Family“ in einer Tabellenähnlichen Struktur gegliedert <p>(spaltenorientierte Datenbanksysteme, 2014)</p>

Tabelle 2.1: wichtigste Kategorien von NoSQL

2.3 R (Programmiersprache)

Laut der R Foundation, welche das Urheberrecht von R besitzt, ist R „a free software environment for statistical computing and graphics“ (R Foundation, 2016). R wurde für Arbeiten im Statistik-Umfeld geschaffen. Aus diesem Grund bringt die Sprache die für statistische Berechnungen notwendige Funktionalität mit und eignet sich, grosse Datensätze zu verarbeiten. Durch die Unterstützung von Packages (Paketen) können aus einem wachsenden Pool neue Funktionen installiert werden, welche von Anwendern erstellt worden sind (siehe Abbildung 2.1). Diese stehen über das „Comprehensive R Archive Network“ (kurz CRAN) zum Download bereit und sind in den meisten Fällen Open-Source. Ausserdem ist R eine Script-Programmiersprache. Als solche ist sie sehr flexibel, da sie nicht kompiliert werden muss. Sie läuft in einem Interpreter, was die Sprache plattformunabhängig macht.

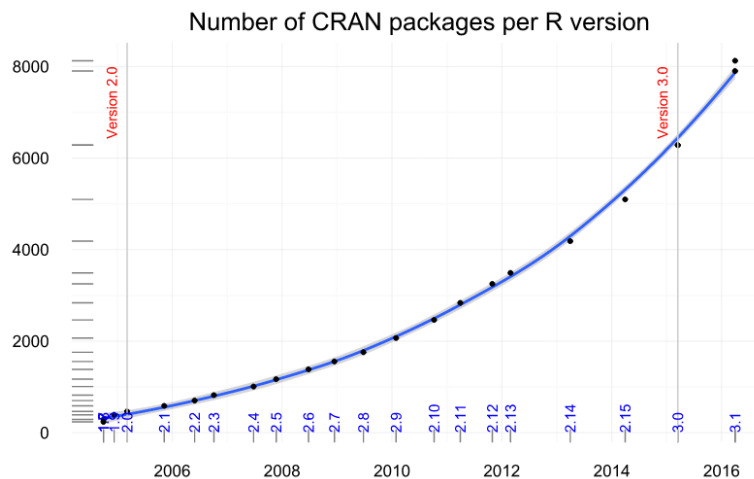


Abbildung 2.1: Anzahl verfügbarer Pakete pro R Version (de Vries, 2016).

2.4 Reaktive Programmierung

Reaktive Programmierung ist ein Paradigma, nach dem Applikationen in einer ereignisorientierten Architektur aufgebaut werden sollen. So können sie die heutigen Anforderungen an Software besser erfüllen, indem sie skalierbarer, zuverlässiger und ausfallsicherer arbeiten. Laut „Reactive Manifest“ müssen reaktive Systeme „stets antwortbereit, widerstandsfähig, elastisch und nachrichtenorientiert sein.“ (Bonér et al., 2014).

Zwar sind alle Benutzeroberflächen zwangsläufig ereignisorientiert, da diese auf Benutzereingaben reagieren müssen. Das Paradigma der reaktiven Programmierung geht aber einen Schritt weiter, in dem eine asynchrone Nachrichtenübermittlung vorausgesetzt wird. Im Gegensatz zu synchronen Programmabläufen

können in asynchroner Verarbeitung mehrere Prozesse aktiv sein (Muehsig, 2009). Solche Applikationen blockieren in der Verwendung nicht, reagieren also stets auf weitere Benutzereingaben.

RStudio beschreibt die Datenverarbeitung folgendermassen: „Reactivity creates the illusion that changes in input values automatically flow to the plots, text, and tables that use the input—and cause them to update.“ (Grolemund, 2015). Das von RStudio entwickelte Shiny Framework gibt dem Benutzer also vor, dass seine Inputs direkte Auswirkungen auf den Output haben. In Realität melden die reaktiven Variablen (zum Beispiel die Variable eines Benutzereingabe-Feld) Änderungen beim Server, worauf dieser die betroffenen Codeabschnitte neu ausführt (siehe Abbildung 2.2).

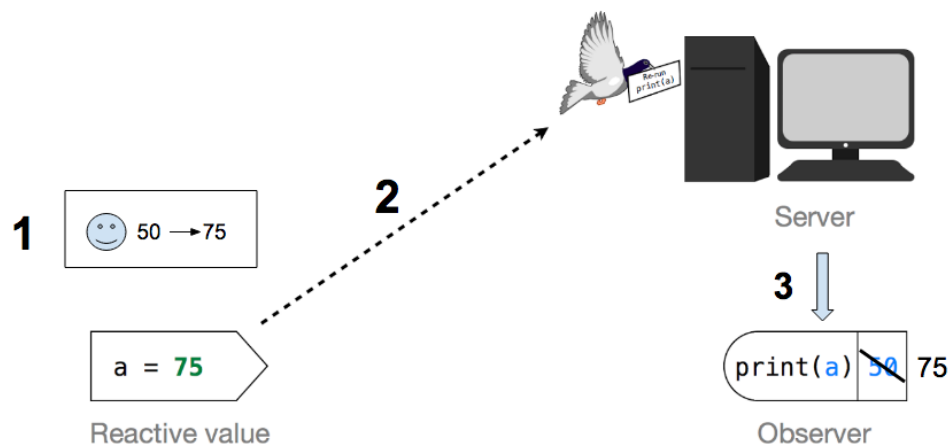


Abbildung 2.2: reaktiver Programmablauf (verändert nach (Grolemund, 2015))

2.5 Shapefile

Ein Shapefile ist ein Datenformat zur Speicherung von Geodaten, welches in Geoinformationssystemen eingesetzt wird. Es besteht nicht aus einer einzelnen Datei, sondern setzt sich aus mehreren Bestandteilen zusammen. Neben den eigentlichen Geodaten können unter anderem Metadaten und Projektionsinformationen als einzelne Dateien darin enthalten sein. Shapefiles unterstützen 3D-Geometrien mit Polygonen, Linien und Punkten. Die Datenstruktur hat sich vor allem wegen ihrer Einfachheit zum Quasi-Standard der GIS-Welt (Geoinformationssystem) durchgesetzt. (ESRI, Inc., 2016).

Kapitel 3

Konzeption

3.1 Vorgehensweise

Das Konzept der Visualisierungsapplikation entstand in einer iterativen Vorgehensweise. Bei jedem Treffen wurden neue Ideen entwickelt, mit welchen die verschiedenen Nutzungsarten ermöglicht werden sollen. Die Anforderungen an die Applikation wurden so stets verfeinert und ausführlicher.

Die iterative Vorgehensweise, nach der diese Bachelorarbeit entstanden ist, ist an das „SCRUM“-Modell angelehnt. Dieses wurde in einer vereinfachten Form angewendet. Der Betreuer Dr. Sinan Teske übernahm dabei die Rolle des Product Owners. Da das Entwicklungsteam nur aus mir bestand, übernahm ich die Rollen des Entwicklungsteams und des SCRUM Masters.

Die durch die SCRUM-Vorgehensweise definierten Elemente wurden je nach Arbeitsschritt eingesetzt. Während der Erarbeitung der Anforderungen wurde kein Backlog verwendet, da sich dieselben Aufgaben immer wiederholten und nicht abgeschlossen werden konnten. Jedoch gab es im Sinne der Sprint-Meetings wöchentlich ein Treffen, um bisherige Arbeitsergebnisse und die Arbeit der nächsten Woche zu besprechen. Dies garantierte, dass Dr. Sinan Teskes Anforderungen korrekt interpretiert und umgesetzt wurden.

In der nächsten Phase, dem Erarbeiten des Datenmodells und der Auswahl der Datenbank, waren die Treffen unregelmässig, dafür aber umso häufiger. So konnten die Fortschritte im Datenmodell anhand der Anforderungen genau überprüft werden.

3.2 Anforderungen an das Datenmodell

Im ersten Schritt soll eine Datenstruktur entworfen werden, mit der die Daten zusammengeführt und gespeichert werden können. Die Datensätze enthalten Informationen zum Energiesystem Schweiz. Es sollen nicht nur Daten des Elektrizitätsnetzes, sondern unter anderem auch des Gas- und Strassennetzes gespeichert werden können. Die Hierarchie der Daten soll erhalten bleiben. Datensätze über das Elektrizitätsnetz sollen also auch als Teil des Energiesystems Schweiz gespeichert werden. Da das Modell zukünftigen Anforderungen standhalten muss, soll es gut skalierbar sein. Ausserdem müssen neue, noch nicht berücksichtigte Datentypen ohne Informationsverlust abgespeichert werden können.

Die bisher berücksichtigten Daten umfassen geographische Daten, Zeitreihendaten und eventbasierte Daten. Geographische Daten entsprechen Punkten, Linien und Polygonen, können also zum Beispiel die Flussverläufe in der Schweiz abbilden. Zeitreihendaten können sowohl normale Zeitreihen, als auch Eigenschaften von Objekten (zum Beispiel Turbinenleistung eines Kraftwerks) darstellen. Eigenschaften werden als Zeitreihen behandelt, da sich ihre Werte ändern können. Die Turbinenleistung zum Beispiel vergrößert sich, wenn ein Kraftwerk ausgebaut wird. Zeitreihen besitzen meistens eine geographische Komponente. So ist zum Beispiel die Stromproduktion eines Kraftwerks mit dessen Standort verbunden. Mit geographischen Daten sollen sich Systemgrenzen darstellen lassen. Dabei entsprechen Polygone den Umrissen von Systemen oder Bilanzräumen. Zusätzlich können geographische Punkte Ein- und Ausgänge für Energie und Stoffe darstellen. Eventbasierte Daten sind Informationen zu bestimmten Zeitpunkten, welche Zeitreihen kommentieren. So kann zum Beispiel der Beginn von Revisionsarbeiten in einem Nuklearkraftwerk gespeichert werden, was Hintergrundinformationen zu Geschehnissen im Datensatz „Produktion Kernkraftwerk Gösgen“ liefert. Eventbasierte Daten können also auf einer Zeitachse dargestellt werden.

Zusätzlich sollen alle Elemente der Datenbank mit Text kommentierbar sein, um Beschreibungen und Arbeitsergebnisse zu speichern. Dabei soll der Name des Erstellers und das Erstellungsdatum gespeichert werden.

Datensätze werden stets von einer Datenquelle bezogen. Diese soll abspeicherbar sein, um Benutzern die Herkunft der Daten nachvollziehbar aufzeigen zu können. Um die Datensätze schneller zu finden, soll es möglich sein, diese zu markieren.

3.3 Anforderungen an das Visualisierungstool

Als zweites Element dieser Bachelorarbeit soll ein Tool zur Erstellung von Visualisierungen konzipiert und implementiert werden, welches auf die Daten der Datenbank zugreift. Zu diesem Zweck soll eine Beispiel-Datenbank erstellt werden, welche alle zukünftig relevanten Arten von Datensätzen enthält.

Da das Erstellen einer aussagekräftigen Visualisierung kein linearer Prozess ist, soll die Applikation als interaktive Webanwendung implementiert werden. Der Benutzer soll also so nach Belieben Filter, Daten und Einstellungen bearbeiten können und sogleich die Auswirkungen auf das Diagramm sehen. Die Visualisierungen sollen zoombar sein und bei Maus-Hover zusätzliche Informationen zu den Datenpunkten anzeigen.

Im Tool sollen zu Beginn die Datensätze für die Visualisierung ausgewählt werden. In einem zweiten Schritt können die Daten nach Datum und Wert gefiltert werden. Danach sollen die geladenen Daten in Plots visualisiert werden, wobei Standardvorgaben benutzt werden sollen. Die generierten Plots soll der Benutzer

bearbeiten können, um ein ideales Layout, eine aussagekräftige Legende und gut ablesbare Daten zu erhalten. Als Visualisierungsarten für Zeitreihendaten sollen mindestens Punkt-, Linien- und Flächendiagramme verfügbar sein. Geographische Daten sollen auf einer Karte dargestellt werden. Um verschiedene Datensätze besser zu vergleichen, soll ein Offset einstellbar sein. Zeitlich versetzte Daten sollen damit überlagert angezeigt werden können. Weiter soll es dem Benutzer möglich sein, Trennlinien zu erfassen. So kann man beispielsweise die Winterzeitumstellung oder einzelne Wochen (als eventbasierte Daten) kennzeichnen. Die geladenen Daten sollen zu temporären, berechneten Feldern zusammengefügt werden können, indem man auf den Datensätzen R Funktionen ausführt. Diese Felder sollen in den Plots angezeigt werden.

Die einzelnen Datensätze sollen in den Visualisierungen farblich voneinander getrennt werden, wobei der Benutzer die Farben bestimmen können soll. Um kohärente Visualisierungen zu erstellen, soll die Farbwahl gespeichert werden und in zukünftigen Plots wiederverwendet werden.

Das Tool soll die geographischen Analysen der Zeitreihen ermöglichen, also Verbindungen zwischen Zeitreihen und geographischen Daten visualisieren. Sind Datensätze mit geographischer Komponente geladen, soll zu diesem Zweck stets zusätzlich eine Karte angezeigt werden. Auf dieser sollen die geladenen Daten ersichtlich sein (als Heatmap oder mit Punkten, Linien und Polygonen). Die zeitliche Dimension soll auf der Karte per Slider oder per Animation visualisiert werden.

In der Applikation soll es die Möglichkeit geben, Kommentare zu aktuell geladenen Datensets, einzelnen Datensätzen oder zu anderen Kommentaren zu erfassen. Sind bereits Kommentare vorhanden, sollen diese während der Arbeit mit den Daten ersichtlich sein.

Um die Plots in Publikationen zu verwenden, sollen diese als Grafiken exportierbar sein. Weiter soll man die geladenen Rohdaten exportieren können. Um Analysen zu ermöglichen, welche in der Applikation noch nicht umgesetzt wurden, soll ein Export als R Code ermöglicht werden. So kann die Visualisierung in einer eigenen R Umgebung weiter angepasst und perfektioniert werden.

Da die Arbeiten an einem Plot einige Zeit in Anspruch nehmen können, soll der Arbeitsstand zwischengespeichert werden können. Dadurch kann die Arbeit an einem noch nicht fertiggestellten Diagramm wiederaufgenommen werden.

Nach Fertigstellung soll die Applikation auf einem Empa-Server bereitgestellt werden. Danach soll sie über das Intranet zur Verfügung stehen.

Kapitel 4

Evaluierung Datenbank

Aus den Anforderungen an die Datenbank lassen sich Anforderungen an das Datenbanksystem ableiten. Diese umfassen:

- **Speicherung der Beziehungen zwischen Daten:** Da die Daten stark vernetzt sind, soll es möglich sein, die Beziehungen zwischen einzelnen Datensätzen in der Datenbank zu speichern.
- **Adaptives Datenmodell:** Neu zu erfassende Daten, welche in den Anforderungen noch nicht berücksichtigt wurden, sollen ohne Informationsverlust eingespeichert werden können.
- **Speicherung geographischer Daten:** Das Datenbanksystem soll geographische Daten und Datenabfragen unterstützen.
- **Performance:** Abfragen sollen performant beantwortet werden.

4.1 Relationale Datenbank

Im Verhältnis zu NoSQL sind relationale Datenbanken deutlich gebräuchlicher. Die meistverwendeten Datenbanksysteme – Oracle, MySQL, Microsoft SQL Server und PostgreSQL – sind alles relationale Datenbanksysteme und teilen etwa zwei Drittel des Datenbankmarktes unter sich auf (DB-Engines Ranking, 2016).

Da wir für unser Projekt hauptsächlich Open-Source-Software einsetzen wollen, kommen von den vier erwähnten Anbietern noch PostgreSQL und MySQL in Frage. Durch die weite Verbreitung der beiden Datenbanken kann in fast jeder gängigen Programmiersprache eine Verbindung zu ihnen hergestellt werden. Beide werden von einer grossen Community getragen, welche in Foren und Blogeinträgen Hilfe bei Problemstellungen bietet.

Relationale Datenbanken können über Verbindungstabellen Beziehungen zwischen Datensätzen speichern. Zur Abfrage werden dann stets Joins notwendig, um Details zu den Verbindungen zu finden. Ein Join verbindet zwei Tabellen zu einer. Dies funktioniert zwar, ist aber weder performant, noch einfach zu handhaben. Da dies eine generelle Beschränkung von klassischen Datenbanken ist, unterscheiden sich MySQL und PostgreSQL in diesem Punkt nicht.

Ein adaptives Datenmodell ist bei beiden Datenbanksystemen über SQL Scripts möglich. Mit ihnen können die Tabellen geändert werden, ohne dass ein Datenverlust entsteht. Weil man aber für wenige Einträge eine neue Spalte einbaut, kann dies über die Zeit die Datenbank aufblasen und unübersichtlich machen.

Mit PostGIS besitzt PostgreSQL eine hervorragende Unterstützung von geographischen Objekten (PostGIS Features, 2016). Auch bei MySQL sind keine Erweiterungen nötig, um die Anforderungen an die geographische Speicherung von Objekten zu erfüllen. (MySQL Reference Book, 2016).

In Untersuchungen zeigt sich, dass „PostgreSQL mit PostGIS ein DBMS ist, welches unter verschiedensten Szenarien eine gute Performance zeigt“ (Santos, Moro, & Davis, 2015). MySQL bietet seit Version 5.7 eine gute Performance: „Auch wenn die Performance in einigen Situationen besser sein könnte, ist sie akzeptabel.“ (Holzgraefe, 2016)

4.2 NoSQL: Graphdatenbank

Die NoSQL-Strömung vereint vier Datenbankarten unter einem Begriff. Von diesen sind Graphdatenbanken am vielversprechendsten, da sich die zu speichernden Daten durch eine starke Vernetzung auszeichnen. Dies ist die Spezialität von Graphdatenbanken

Duden definiert Graphen als eine „grafische Darstellung (z. B. von Relationen) in Form von [markierten] Knoten[punkten] und verbindenden Linien (Kanten)“ (Duden, 2016). Datenbanken, welche Daten als Graph abspeichern, sind also darauf spezialisiert, Verknüpfungen zu speichern. Laut dem Datenbank-Ranking von DB-Engines Ranking ist Neo4j die populärste Datenbank dieses Typs (DB-Engines Ranking, 2016). Da die Software ausserdem als Open Source veröffentlicht ist (Neo Technology, Inc., 2016a), wird dieser Datenbankanbieter evaluiert.

Neo4j speichert Daten als Nodes (Knoten) mit Properties (Eigenschaften) ab. Den Nodes kann man Labels zuteilen, welche den Typ der Daten festhalten. Ein grosser Unterschied ist jedoch, dass sich Nodes trotz gleichem Label in den Properties unterscheiden können. Dieser Umstand ist vorteilhaft, wenn neue, im Datenmodell noch nicht berücksichtigte Daten abgespeichert werden sollen. Bringen diese Zusatzinformationen mit, können sie trotz anderer Form mit demselben Label versehen werden. So wird Informationsverlust vermieden.

Geographische Funktionen sind bei Neo4j nicht im Lieferumfang. Da aber Erweiterungen unterstützt werden (Neo Technology, Inc., 2016b), sind die Funktionalitäten mit dem „Neo4j Spatial Plugin“ nachrüstbar. Zusammen mit dem Plugin erfüllt Neo4j alle Anforderungen in Bezug auf den Umgang mit geographischen Daten (GitHub, 2016).

Die Performance ist ähnlich wie PostgreSQL mit PostGIS (Santos, Moro, & Davis, 2015). Somit sind alle evaluierten Datenbanken auf demselben Performance-Level.

4.3 Auswertung

Anforderung	Relationale Datenbank	Graphdatenbank
Speicherung der Beziehungen	-	++
Adaptives Datenmodell	-	++
GIS-Daten speichern	++	+
Performance	++	++

-- nicht erfüllt / - schlecht erfüllt / + gut erfüllt / ++ sehr gut erfüllt

Tabelle 4.1: Subjektive Bewertung der Erfüllung der Anforderungen

Da sich die Energiedaten und deren Verbindungen ohne Informationsverlust als Graph abgebildet lassen, können diese gut in einer Graphdatenbank gespeichert werden. Mit der Open-Source Lösung Neo4j gibt es dafür einen ausgezeichneten Datenbankanbieter. Es ist nicht nur die am weitesten verbreitete Datenbank dieses Typs, sondern auch praxiserprobt durch Kunden in diversen Wirtschaftszweigen (Neo Technology, Inc., 2016c). Dank den Eigenschaften von Neo4j ist das Datenmodell beliebig erweiterbar, ohne dass bestehende Datensätze geändert werden müssen. Für die Arbeit mit Geo-Informationen wird Neo4j mit dem Plugin Neo4j Spatial erweitert. Aus diesen Gründen hat die Evaluation ergeben, dass Graphdatenbanken die Anforderungen besser als relationale Datenbanken erfüllen.

Kapitel 5

Evaluierung Visualisierungstool

Aus den Anforderungen an die Gesamtlösung lassen sich die Anforderungen an das Visualisierungstool ableiten. Diese umfassen:

- **Kompatibilität mit gewählter Datenbank:** Das Visualisierungstool soll sich an Neo4j anbinden lassen.
- **Nichtlinearer Programmablauf / Interaktivität:** Das Visualisierungstool soll keinen starren Ablauf haben. Es soll also möglich sein, Daten nachzuladen, Daten zu löschen und die Visualisierung anzupassen
- **Umfassende Visualisierungsmöglichkeiten:** Das Tool soll für alle Datenarten eine geeignete Visualisierungsmöglichkeit bieten.
- **Kartendarstellung:** Neben den Plots für Zeitreihendaten soll eine Kartendarstellung für Geo-Daten und Datensätze mit Lokalisierung möglich sein.
- **Exportmöglichkeiten:** Um die Plots in einer Präsentation verwenden zu können, müssen Exportmöglichkeiten bereitstehen. Es muss mindestens ein Bildexport verfügbar sein.
- **Berechnete Felder:** Datensätze sollen mit Funktionen temporär verändert werden können, um so neue Erkenntnisse zu gewinnen.
- **Möglichkeit zur Verwaltung der Datenbank:** Vorhandene Datensätze sollen kommentierbar sein. Neue Datensätze (zum Beispiel aus berechneten Feldern) soll man abspeichern können.
- **Data Exploring:** Es soll möglich sein, die in der Datenbank vorhandenen Daten durchzusehen. Zusammenhänge zwischen Datensätzen sollen ersichtlich sein, um diese analysieren zu können.
- **Arbeitsstand speichern:** Da die Arbeit an Visualisierungen einige Zeit in Anspruch nehmen kann, soll der Arbeitsstand zwischengespeichert werden können.
- **Quelloffenheit / Lizenzkosten:** Das Benutzen von Open Source Software spart Kosten und verringert Abhängigkeiten eines Software-Herstellers (Bridge, 2016). Ist ein Programm nicht quelloffen, sollten die Lizenzkosten tief gehalten werden.

5.1 bestehende Visualisierungstools

5.1.1 Tableau Desktop

Tableau Desktop ist ein Visualisierungstool von Tableau Software, einem börsennotierten Unternehmen welches langjährige Erfahrung in Business Intelligence besitzt. Es bietet „Analysen für alle“ und will den Benutzern helfen, möglichst schnell Erkenntnisse zu gewinnen (Tableau Software, 2016a).

Leider gibt es keinen nativen Datenbankanschluss an Neo4j. Mit einem Workaround kann jedoch trotzdem eine Verbindung hergestellt werden, nämlich über einen Tableau Web Connector (Becher, 2016). Dieser wird jedoch nicht offiziell unterstützt. Die Erstellung eines Plots in Tableau ist komplett interaktiv. Auch das Nachladen von Daten über den Web Connector ist möglich. Durch die langjährige Erfahrung des Herstellers sind zahlreiche Visualisierungsarten für alle Daten vorhanden (Tableau Software, 2016b). Die Exportmöglichkeiten genügen den Anforderungen knapp. Auch eine Kartendarstellung ist möglich, diese unterstützt jedoch kein WKT, das Standard-Speicherformat von Neo4j Spatial (Tableau Community, 2014). Karten mit geographischen Daten aus Neo4j können nicht erstellt werden, da keine Möglichkeit zur Echtzeitkonvertierung existiert. Das Erstellen von berechneten Feldern ist möglich. Da der Web Connector grundsätzlich nur zum Auslesen aus der Datenbank entwickelt wurde, kann die Datenbank nicht darüber verwaltet werden. Es ist also weder möglich, berechnete Daten abzuspeichern, noch Kommentare zu erfassen. Gleiches gilt für das erforschen der Datenbank: Dafür müsste ein separates Tool zur Verfügung stehen. Der Arbeitsstand kann als Datei gespeichert werden. Dieser kann sogar mit einem Gratis-Reader geöffnet werden (Tableau Software, 2016c). Tableau ist keine quelloffene Software. Die Lizenzkosten sind auch für eine offline Version hoch (Tableau Software, 2016a).

5.1.2 Microsoft Power BI

Power BI ist eine Visualisierungsanwendung für den Business Intelligence Bereich, welche von Microsoft vertrieben wird. Mit der Integration von Cloud und Mobilgeräten bietet die Software hohe Benutzerfreundlichkeit und unterstützt viele Endgeräte (Microsoft, 2016a).

Auch Power BI bietet keine native Neo4j Unterstützung. Es stehen allerdings ein R Connector und ein Web Connector zur Verfügung, mit dem sich eine Verbindung aufbauen lässt (Iseminger, 2016a). Power BI unterstützt zwar eine interaktive Gestaltung von Plots, unterstützt aber das Nachladen von Datensätzen aus der Datenbank nur halbwegs. Grundsätzlich wäre die Möglichkeit vorhanden, aber

wegen umständlicher Bedienung ist es keine praxistaugliche Lösung. Obwohl die Software vergleichsweise neu auf dem Markt ist, bietet sie viele Visualisierungsmöglichkeiten, welche über einen Downloadbereich „Custom Visuals Gallery“ erweitert werden können (Microsoft, 2016b). Dort ist die „Enhanced Map with Custom Geography“ erhältlich, welche eine Kartendarstellung von WKT-Daten ermöglicht (deldersveld, 2016). Exportmöglichkeiten, wie Export als Bild, sind reichhaltig vorhanden. Durch die Integration in Microsoft Office können Daten in andere Office Programme wie Excel und PowerPoint exportiert werden. Berechnete Felder sind grundsätzlich möglich, allerdings werden nur sogenannte DAX (Data Analysis Expression) Funktionen unterstützt, welche Endanwender von Excel kennen (Iseminger, 2016b). Wie Tableau unterstützt Power BI das Schreiben in die Datenbank nicht. Somit können keine Veränderungen vorgenommen werden. Dasselbe gilt für Data Exploring, auch hier wäre ein eigenes Tool notwendig. Der Arbeitsstand kann ähnlich wie bei Tableau in eine Datei gespeichert werden. Hier zeigt sich aber eine weitere Einschränkung: Die Gesamtgrösse aller geladener Daten darf 250 Megabytes nicht überschreiten (Saxton, 2015). Obwohl Power BI nicht quelloffen ist, wird die Applikation in einer Basisvariante gratis vertrieben. Die kostenpflichtige Pro-Version bietet bezogen auf die Anforderungen keine Vorteile.

5.1.3 Qlik QlikView

Wie die beiden erstgenannten Applikationen ist QlikView ein Visualisierungstool, welches sich auf Business Intelligence spezialisiert hat. Die Software ermöglicht es, eigene Analyseapps zu erstellen. Diese sollen in Betrieben für neue Erkenntnisse sorgen (Qlik, 2016a).

QlikView unterstützt das von Neo4j angebotene JDBC (Java Database Connectivity) durch ein Plugin, welches im Qlik Market heruntergeladen werden kann (TIQ Solutions, 2016). QlikView unterstützt die interaktive Gestaltung von Visualisierungen. Doch obwohl es technisch möglich wäre, konnte über das Plugin kein Datensatz nachgeladen werden. Die Benutzung der Applikation ist somit nicht vollständig interaktiv. Positiv ist, dass alle benötigten Visualisierungsarten unterstützt werden, weitere können im Market heruntergeladen werden. Die verfügbare Kartendarstellung bietet vollen WKT-Support (Qlik, 2016b). Exportmöglichkeiten sind vorhanden, jedoch ist QlikView darauf ausgelegt, als App benutzt zu werden. So sind Export nach PDF und PNG mühsamer als mit Konkurrenzprodukten, dafür kann man die gesamte Analyse als Datei abspeichern und mit anderen teilen. Diese können die Datei in einem kostenlosen Reader öffnen und betrachten. Das Erstellen von berechneten Feldern wird unterstützt. Auch QlikView kann die Datenbank nicht beschreiben, somit können die Daten nicht verändert oder kommentiert werden. Datenexploration wäre so kaum möglich. Wie erwähnt

kann der Arbeitsstand in eine Datei gespeichert werden. QlikView ist nicht quell-offen, aber in einer Personal Edition gratis erhältlich. Diese ist an zwei Orten limitiert: Erstens unterstützt sie das Teilen der Dateien nicht, zweitens können keine Templates benutzt werden (Qlik, 2013).

5.1.4 Eigenentwicklung

Standardsoftware-Programme, wie in den Abschnitten 5.1.1 bis 5.1.3 beschrieben, decken schon mit den mitgelieferten Funktionalitäten die Anforderungen an Visualisierungen ab. Implementiert man eine eigene Lösung, müssen diese Funktionen selbst implementiert werden. Daraus resultiert ein im Vergleich zu bestehenden Lösungen deutlich grösserer Aufwand, welcher mit jeder zusätzlichen Funktion wächst. Dafür können Funktionen direkt aufgrund der eigenen Ansprüche implementiert werden. Somit können alle Anforderungen perfekt abgedeckt werden. Da keine unnötigen Funktionen implementiert werden, sind Eigenentwicklungen zudem einfacher zu bedienen und übersichtlicher. Es gilt stets abzuwägen, ob sich der Mehraufwand einer eigenen Implementation lohnt. Um zu zeigen, wie eine Eigenentwicklung aussehen könnte, wurde ein Entwurf erstellt (siehe Abbildung 5.1)

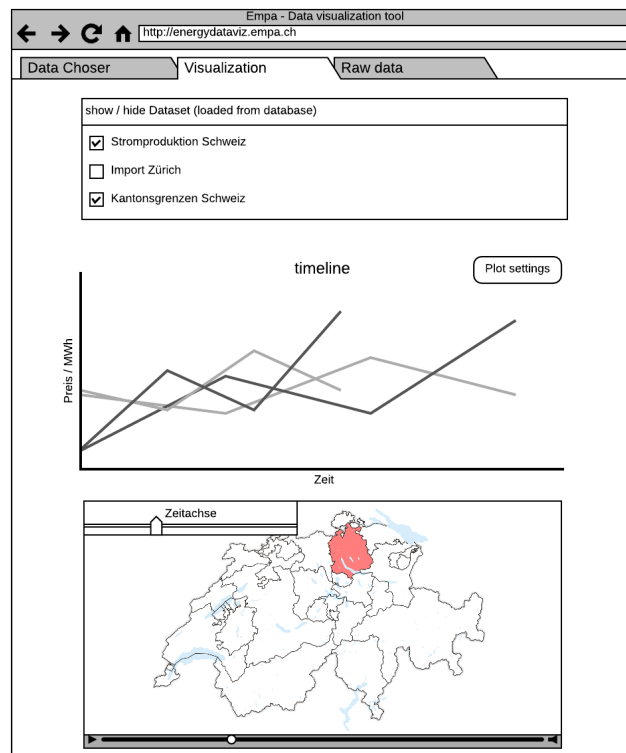


Abbildung 5.1: Entwurf Eigenentwicklung

5.2 Auswertung

Anforderung	Tableau Desktop	Microsoft Power BI	Qlik QlikView	Eigenentwicklung
Kompatibilität mit Datenbank	-	--	--	++
Interaktivität	++	+	+	++
Umfassende Visualisierungsmöglichkeiten	++	++	++	++
Kartendarstellung	-	+	++	++
Exportmöglichkeiten	+	++	-	++
Berechnete Felder	+	--	++	++
Datenbankverwaltung	-	--	--	++
Data Exploring	-	--	--	++
Arbeitsstand speichern	++	--	++	++
Quelloffenheit und Kosten	--	+	+	++

-- nicht erfüllt / - schlecht erfüllt / + gut erfüllt / ++ sehr gut erfüllt

Tabelle 5.1: Subjektive Bewertung der Erfüllung der Anforderungen

Keines der getesteten bestehenden Visualisierungslösungen konnte alle Anforderungen erfüllen. Aus diesem Grund fällt die Entscheidung auf die Eigenentwicklung. Der Mehraufwand ist in diesem Fall gerechtfertigt, da die Applikation auf das Datenmodell abgestimmt werden kann. In Zusammenhang mit dem Datenmodell können die Anforderungen also optimal umgesetzt werden.

Kapitel 6

Umsetzung

6.1 Vorgehensweise

In der Konzeption wurden die Anforderungen in zwei Teilbereiche aufgeteilt, nämlich Datenbank und Visualisierungstool. Da das Visualisierungstool auf der Datenbank aufbaut, wurde zuerst das Datenmodell entworfen (siehe Abschnitt 6.2). Nachdem eine erste Version davon fertiggestellt war, begannen die Arbeiten am Visualisierungstool. Hier wurde zuerst ein Prototyp erstellt, um die vorgesehenen Technologien auszutesten (siehe Abschnitt 6.3). Erst nach erfolgreicher Prüfung begannen die Arbeiten am eigentlichen Tool (siehe Abschnitt 6.4).

In der Konzeptionsphase wurde in dieser Bachelorarbeit nach einer abgewandelten SCRUM-Methode vorgegangen, in der einige Elemente weggelassen wurden. Während der Implementation des Visualisierungstools verwendeten wir nun alle SCRUM-Mittel. Als Backlog und Sprint-Planning Tool wurde www.trello.com verwendet (Trello, Inc., 2016). Dieses bietet auf einer Web-Oberfläche die Möglichkeit Listen zu erstellen, welche mit Karten befüllt werden können (siehe Abbildung 6.1). Karten stellen einzelne Aufgaben dar, welche mit Drag & Drop zwischen den Listen hin und her geschoben werden. In der Sprint Planning Besprechung wurde entschieden, welche Anforderungen als nächstes erfüllt werden sollen. Zusammen mit der Aufwands-Schätzung wurden sie im Backlog platziert. Da Aufgaben von Liste zu Liste verschoben wurden, konnte stets nachvollzogen werden, wie weit fortgeschritten die Arbeit an jeder Aufgabe war.

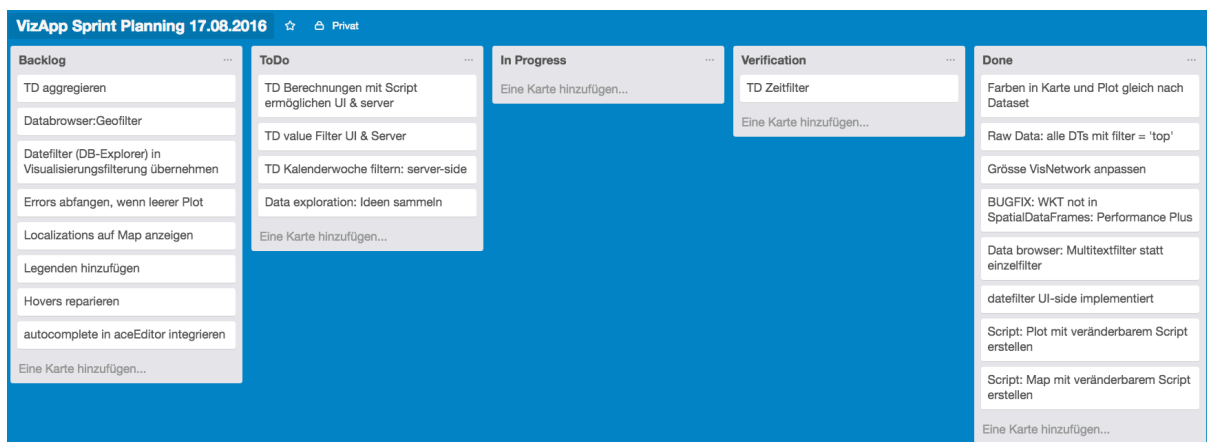


Abbildung 6.1: Trello Board eines Sprints

6.2 Datenmodell

Als Graphdatenbank verlangt Neo4j nach einem komplett anderen Datenmodell, wie man dies in einer relationalen Datenbank einsetzen könnte. Als Hilfestellung in der Datenmodellierung stellt Neo4j einen Leitfaden „Graph Data Modeling Guidelines“ (Neo Technology, Inc., 2016d) zur Verfügung, welcher die notwendigen Techniken aufzeigt und erklärt.

Energiedaten unterscheiden sich in Form und Auflösung stark. Dieser Umstand erschwerte die Entwicklung eines Datenmodells. Durch die iterative Vorgehensweise wurde das Datenmodell aber stets verbessert und um neue Datenarten erweitert. Das Resultat ist in Abbildung 6.2 ersichtlich. Das dort abgebildete Datenmodell kann nun alle vorgesehenen Daten aufnehmen.

Im Zentrum steht der Objekttyp „Dataset“ (Datensatz), welcher für jeden in der Datenbank abgebildeten Datensatz erstellt wird. Im rechten Bereich der Grafik stehen die Data Nodes („SpatialData“, „TemporalData“ und „EventData“). Objekte dieser Typen stellen einzelne Datenpunkte dar und gehören stets zu einem Dataset. Die restlichen Objekttypen werden benutzt, um Datasets zu beschreiben (siehe Abschnitt 6.2.1 Beschreibung der Objekttypen).

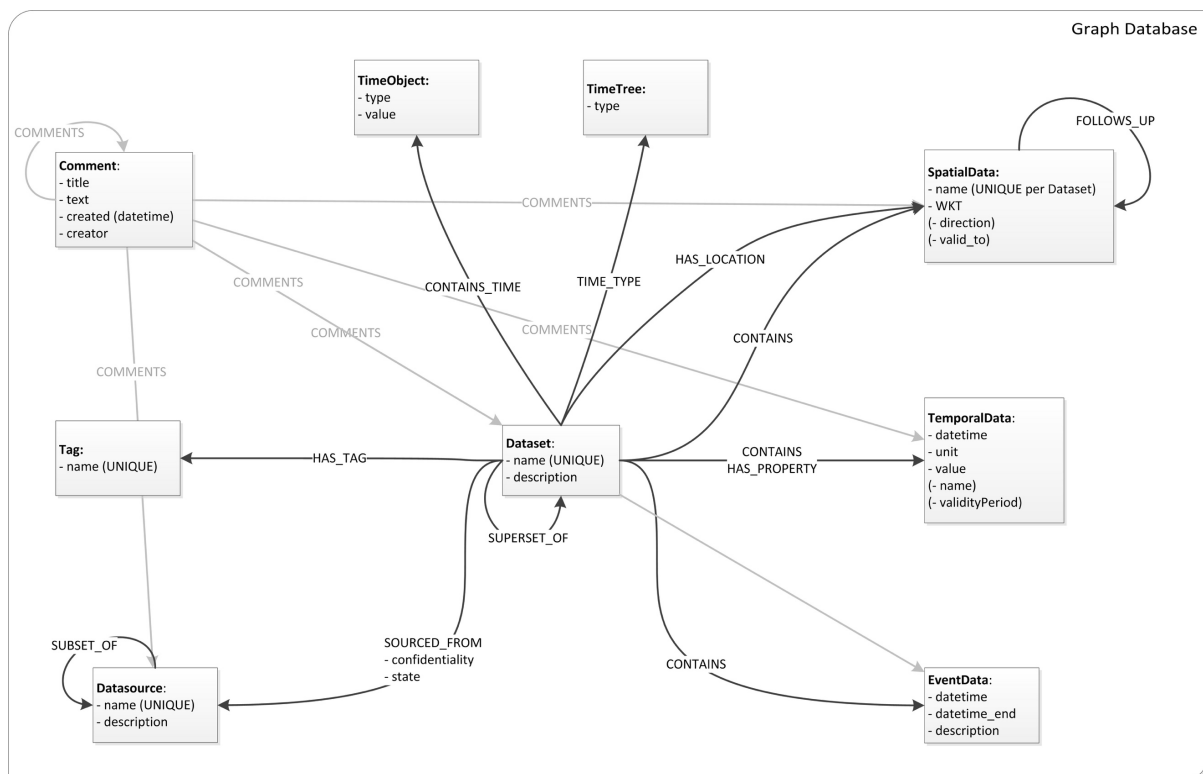


Abbildung 6.2: Datenmodell Graphdatenbank

6.2.1 Beschreibung der Objekttypen

SpatialData sind Objekte, welche geographische Daten speichern. Diese haben einen im Datenset einzigartigen Namen, um sie zu identifizieren. Das Feld „WKT“ (Well-Known Text) ist ein Textfeld, dessen Inhalt ein geographisches Objekt definiert. Das Objekt kann mit dem Feld „direction“ erweitert werden. Dieses ist notwendig, wenn ein Punkt eine Systemgrenze beschreibt. In diesem Fall soll es in einer Visualisierung möglich sein, den Punkt als Pfeil darstellen zu können. Das Feld „valid_to“ erweitert SpatialData Objekte, indem ein zeitlicher Gültigkeitsbereich definiert werden kann. Fusionieren zum Beispiel zwei Schweizer Gemeinden, soll die neue Gemeinde als geographisches Objekt hinzugefügt und die beiden alten für ungültig erklärt werden.

TemporalData sind Zeitreihenobjekte, wobei jedes Objekt ein Messwert darstellt. Diese beschreiben einen Zeitpunkt („datetime“) mit einem Wert („value“), welcher in einer festgelegten Einheit („unit“) dargestellt ist. Neben Messreihen können mit TemporalData Objekten auch „Properties“ (Eigenschaften) eines Datensatzes abgebildet werden. Die Felder „name“ und „validityPeriod“ werden in diesem Anwendungsfall benötigt, damit den Eigenschaften einen zeitlichen Gültigkeitsbereich und eine Beschreibung zugewiesen werden können.

EventData sind Ereignisse und können Zeitreihen kommentieren, indem sie die Zeitachse in Abschnitte einteilen (wie zum Beispiel Sommer- / Winterzeit) oder einen Zeitpunkt beschreiben (wie zum Beispiel ein Hochwasserereignis). Sie sind durch Start- und Endzeitpunkt („datetime“ und „datetime_end“) sowie eine Beschreibung „description“ definiert. Punktuelle Ereignisse können dargestellt werden, indem das Feld „datetime_end“ leergelassen wird.

Dataset (Datensatz) Objekte sind die zentralen Elemente des Datenmodells. Sie strukturieren die Datenpunkte über die „CONTAINS“-Relation in einem Containerformat und ermöglichen die Beschreibung der Daten durch die Felder „name“ und „description“. Die Relation „HAS_PROPERTY“ erlaubt die Erfassung von Eigenschaften, welche in Zahlenwerten ausgedrückt werden können (z. B. die Eigenschaft „Turbinenleistung“ eines Kraftwerkes). Enthält ein Datensatz ortsbezogene Informationen, wird dies mit einer „HAS_LOCALIZATION“ Verknüpfung zum jeweiligen SpatialData Objekt erfasst. Über „SUPERSET_OF“ können Supersets gebildet werden, welche ein oder mehrere Datensätze enthalten. So können Beziehungen zu anderen Datensets dargestellt werden.

Tags ermöglichen die Markierung von Datensätzen. So können beispielsweise alle in einem Paper verwendeten Datensätze über das Feld „name“ des Tags wiedergefunden werden.

Datasource sind Objekte, welche den Ursprungsort der Daten beschreiben. Datasources sind definiert durch „name“ und „description“ und können hierarchisch

geordnet sein (beispielsweise ist die Quelle „Herr Müller von Swissgrid“ ein Teil der Datenquelle „Swissgrid AG“). Die Relation „SOURCED_FROM“, welche Datensatz mit Datenquelle verbindet, enthält die Eigenschaft „confidentiality“. Mit dieser können Datensätze als vertraulich eingestuft werden.

Comments können beliebige Objekte der Datenbank kommentieren. So können wichtige Erkenntnisse, Unklarheiten, spezielle Gegebenheiten und Ähnliches beschrieben werden. Ein Kommentar besteht aus einem Titel („title“) und einem Text („text“) und wird mit Erstellungszeitpunkt („created“) und Erfasser („creator“) markiert.

TimeObjects sind Objekte, welche zur Beschleunigung des Filterns eingesetzt werden. Diese Objekte umfassen Wochen mit Wochennummern (z. B. „type“ = „week“ und „value“ = 52), Jahre mit Jahreszahlen („year“, „2015“) sowie Wochentage („weekday“, „Monday“). Datensätze werden mit den jeweiligen TimeObjects markiert, um zum Beispiel die Suche nach spezifischen Jahren oder Wochen zu beschleunigen. Dies wäre auch über das Filtern der datetime Eigenschaften der Objekte möglich. Aufgrund der hohen Zahl an notwendigen Datenbankzugriffen genügte jedoch die Performance den Anforderungen nicht.

TimeTree Objekte definieren, in welchem zeitlichen Abstand die Messungen erfolgt sind (in der Eigenschaft „type“). TimeTree Objekte werden in einen Baum gegliedert (siehe Abbildung 6.3).

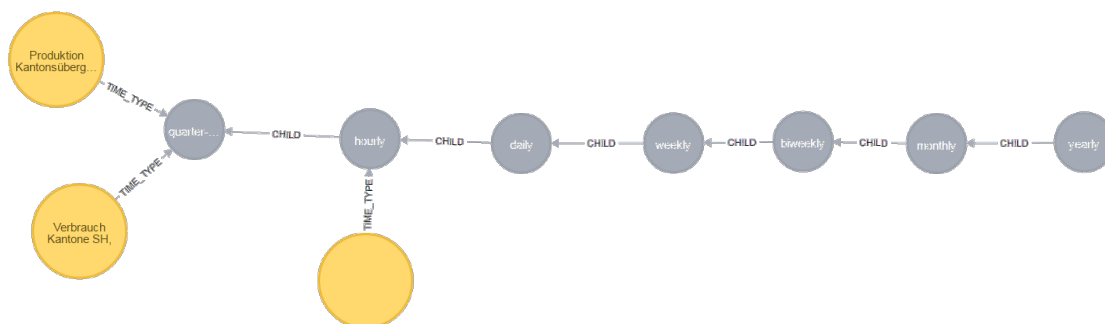


Abbildung 6.3: TimeTree Baum mit Beispieldatensätzen

6.2.2 Aufbau Beispieldatenbank

Um das Datenmodell ausführlich testen zu können, wurde eine Beispiel-Datenbank erstellt. Diese enthält jede Datenart, welche in den Anforderungen spezifiziert wurde. Damit die Datenbank schnell aufgesetzt werden kann, wurde ein R Script entwickelt, welches die Beispiel-Daten automatisch in eine neue Datenbank einfügt (SetupExampleDB.R).

Die Beispieldatenbank enthält die nachfolgend aufgeführten Datensätze. Zeitreihendaten wurden jeweils mit den TimeObjects und dem TimeTree verbunden.

- Netzdaten 2015 mit 64 Datensätzen, 15-Minütlich, Datenquelle Swissgrid
- Landesgrenze Schweiz als Polygon, Datenquelle Swisstopo
- Kantons Grenzen Schweiz als Polygon, Swisstopo
- Superset der Landes- und Kantons Grenzen
- Positionen der Abwasserreinigungsanlagen Schweiz als Punkte, Swisstopo
- Turbinenleistung des Kernkraftwerk Gösgen als Eigenschaft („HAS_PROPERTY“ Relation)
- Sommerzeitstart und –ende als Eventdaten
- Einige Kommentare
- Einige Tags

```
# Create a sample superset -----
new.superset <- importDataset(graph, "Grenzverläufe Regionen Schweiz",
                              "Grenzverläufe der verschiedenen schweizer Regionen")

# Get Kantons- und Landesgrenzen Schweiz put them into new.superset
borders <- getNodes(graph, "MATCH (n:Dataset) WHERE n.name IN ['Landesgrenze Schweiz',
                                                             'Kantons Grenzen Schweiz'] RETURN n")

importDatasetToDs(graph, new.superset, borders)

# Tag a node -----
createTagOnNode(graph, subset[[1]], "mit QGIS erstellt")

# Create Event Data -----
sommerzeit_ds <- importDataset(graph, "Sommerzeit",
                              "Sommerzeitbeginn und -ende")

start <- ymd_hm("2015-03-29 03:00", tz = "Europe/Berlin")
end <- ymd_hm("2015-10-25 02:00", tz = "Europe/Berlin")
event <- createEventData(graph, start, end,
                        description = "Sommerzeit CET / CEST")

connectDataToDs(graph, sommerzeit_ds, event)
```

Programmcodex 6.1: Ausschnitt aus dem Script, welches Beispieldaten importiert

6.3 Prototyp Stromerzeugung

Bevor mit der Implementation des Visualisierungstools begonnen wurde, sollten die einzusetzenden Technologien (siehe Abschnitt 6.4.1) in einem Prototyp geprüft werden. Dies sollte zeigen, dass die Anforderungen erfüllt werden können. Der Prototyp baut auf einer bestehenden Applikation (siehe Abbildung 6.4) auf, welche anhand Benutzereingaben Zeitreihencharts und Karten mit Daten des Schweizer Elektrizitätsnetzes zeichnete. Das ursprüngliche Tool wurde mit Python entwickelt.

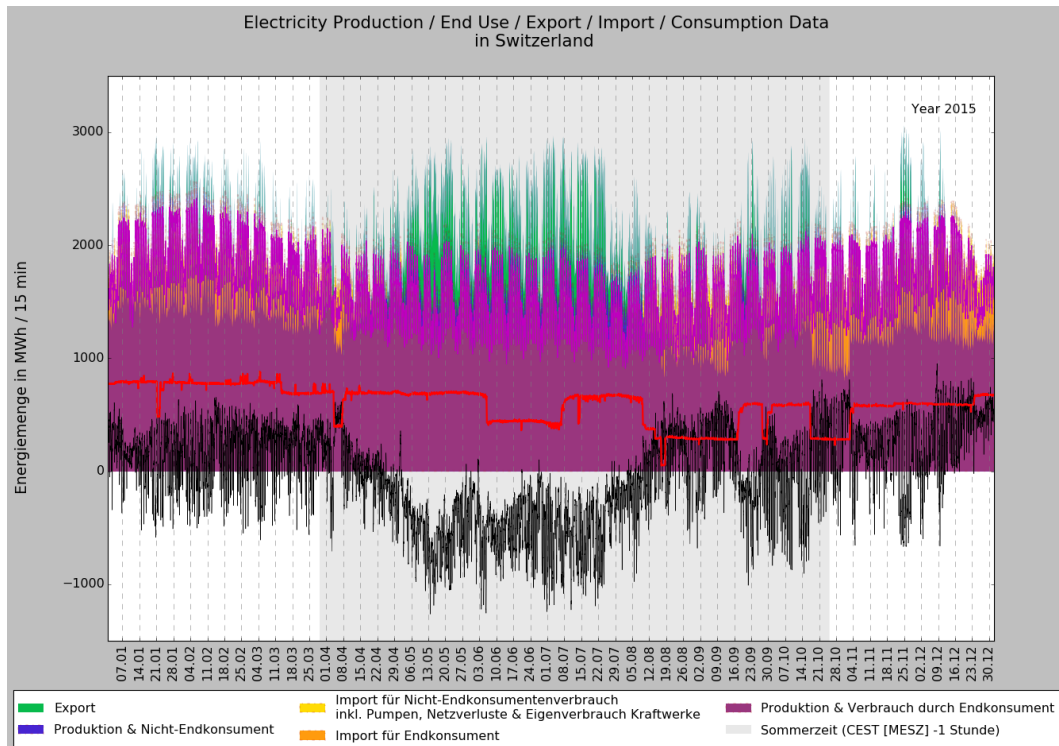


Abbildung 6.4: Beispiel einer Zeitreihe des Vorbilds

Der Prototyp ist in Abbildung 6.5 ersichtlich. Wie in der ursprünglichen Applikation kann der Benutzer Daten als Linien und Flächen zur Visualisierung hinzufügen. Zusätzlich ist ein Datumsfilter verfügbar, mit welchem das Start- und Enddatum festgelegt werden kann. Im Gegensatz zum Python Tool ist die Applikation interaktiv und muss nicht über einer Konfigurationsdatei bedient werden. Da die Anforderungen an den Prototyp erfolgreich umgesetzt werden konnten, wurden die eingesetzten Technologien nicht verändert.

SwissGrid Energy Data Visualization

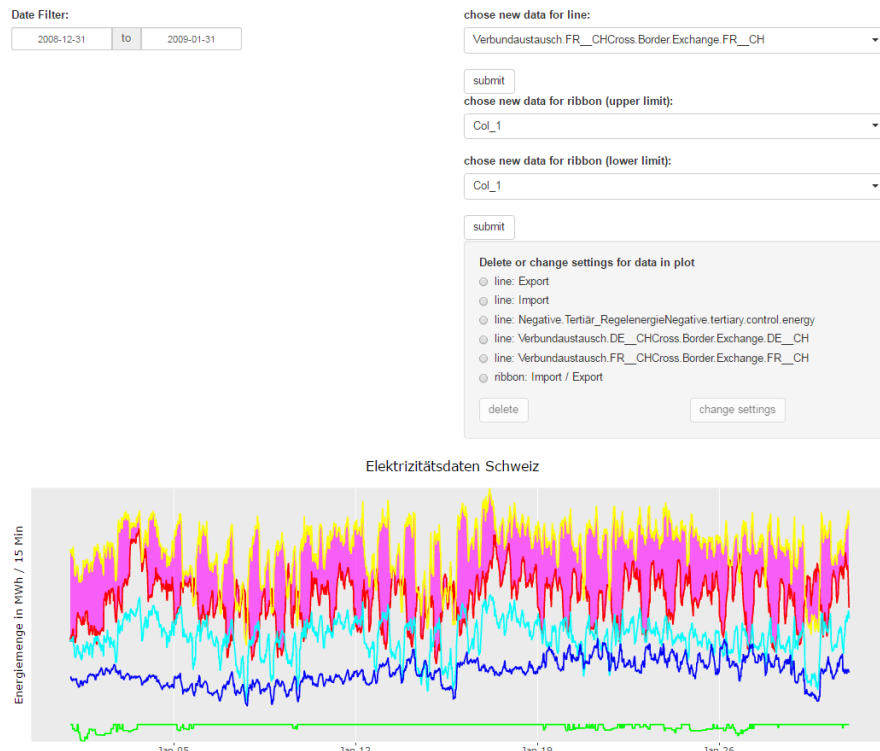


Abbildung 6.5: Prototyp interaktive Shiny Applikation

6.3 Visualisierungstool

Das Visualisierungstool ist in vier Tabs gegliedert, welche sich an den Arbeitsschritten bei der Erstellung einer Visualisierung orientieren (siehe Abbildung 6.6):

- Tab Data Overview
- Tab Data Explorer
- Tab Visualization
- Tab Raw Data

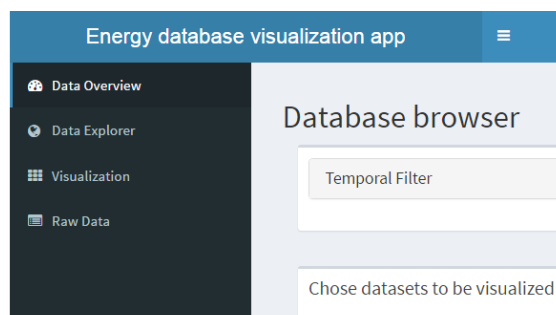


Abbildung 6.6: Übersicht der Tabs des Visualisierungstools

Aufgrund der grossen Zahl an Datensätzen können die Daten nicht vollständig in den Arbeitsspeicher geladen werden. Im Tab Database Overview wird dem Benutzer stattdessen eine Übersicht über die Datenbank angezeigt. Hier kann er Datensätze auswählen, welche im Anschluss in den weiteren Tabs zur Verfügung stehen. Der Tab Data Explorer bietet Funktionen, um nach verwandten Datensets zu suchen. Mittels fünf Suchmethoden werden dem Benutzer Datensets angezeigt, welche für seine Visualisierungen relevant sein könnten. Im Tab Visualization werden die Visualisierungen generiert. Diese können verändert und exportiert werden. Abschliessend zeigt der Tab Raw Data die geladenen Rohdaten. So können Ausreisser nachvollzogen werden.

Anhang A zeigt Screenshots aller Tabs in voller Grösse. Im Anhang B ist ein Benutzerhandbuch zu finden. Dieses führt aus, wie die Applikation benutzt werden und welche Funktionen zur Verfügung stehen.

Abbildung 6.7 zeigt den strukturellen Aufbau des Visualisierungstools inklusive Datenbankanbindung. Darin sind die eingesetzten Technologien zusehen, welche in Abschnitt 6.3.1 näher behandelt werden.

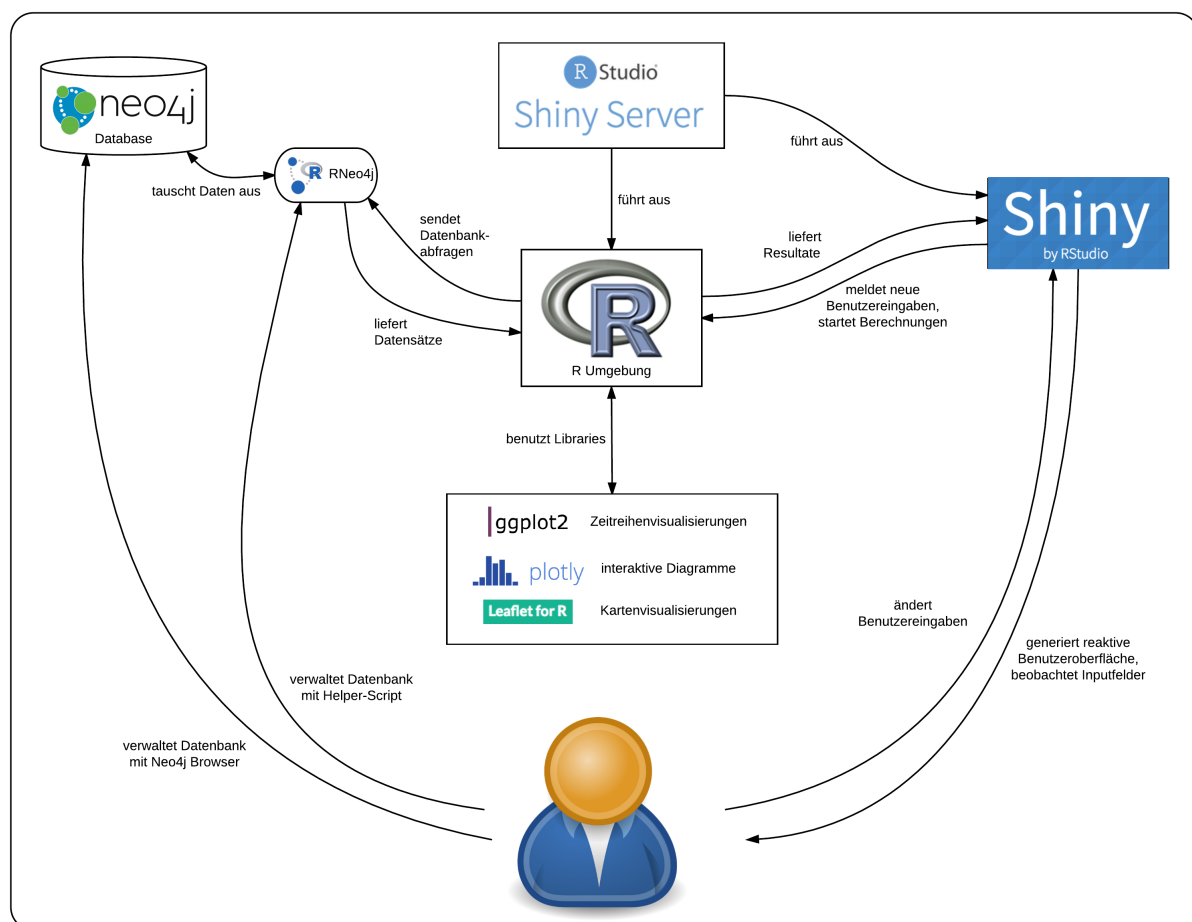


Abbildung 6.7: Struktureller Aufbau der Applikation mit eingesetzten Technologien

6.3.1 eingesetzte Technologien

Die Entwicklungsarbeit erfolgte mit der Programmiersprache R, diese stellt somit das Kernelement der Applikation dar. Die Sprache wurde für Arbeiten im Statistik-Umfeld geschaffen. Aus diesem Grund bringt sie die für statistische Berechnungen notwendige Funktionalität mit und eignet sich, grosse Datensätze zu verarbeiten. Ausserdem sind Packages verfügbar, mit welchen Visualisierungen erzeugt werden können. Als eine der laut IEEE Spectrum Magazin am weitesten verbreiteten Programmiersprachen (Diakopoulos & Cass, 2016) hat R eine grosse und sehr aktive Community, welche Unterstützung und Hilfe bietet. Als Beispiel sei an dieser Stelle die Seite r-bloggers (r-bloggers, 2016) genannt, auf der täglich eine Vielzahl an Blogeinträgen die neusten Entwicklungen, neue Tutorials oder einfach spannende Beispiele einzelner Packages zeigen. Als quelloffene Sprache wird sie laufend weiterentwickelt und verbessert.

Für die Anbindung an Neo4j wurde von Neo4j, Inc. das Paket „RNeo4j“ entwickelt (White, 2016). Durch die native Anbindung kann die Datenbank sowohl gelesen als auch beschrieben werden.

Ein weiteres für R verfügbares Paket ist das Framework „Shiny“, welches die Erstellung interaktiver Webapps mit wenig Aufwand ermöglicht. Es baut dabei auf dem Grundsatz der reaktiven Programmierung auf. Shiny unterstützt sämtliche Bildschirmgrössen durch das Einbinden von Bootstrap, einem Framework, welches HTML-Elemente zur Verfügung stellt. Diese skalieren mit der Bildschirmgrösse, sodass damit erstellte Webseiten auf sämtlichen Geräten korrekt dargestellt werden. Die Herausgeber, RStudio, Inc., bieten mit Shiny Server überdies eine quelloffene Applikation an, um Shiny Apps über das Netzwerk oder Internet zur Verfügung zu stellen (RStudio, 2016).

Als Pakete zur Datenvisualisierungen können „ggplot2“ (Hadley, 2013), „Plotly“ (plotly, 2016) und „Leaflet“ (Leaflet for R, 2015) eingesetzt werden. Ggplot2 und Plotly ermöglichen zusammen die Erstellung von interaktiven Zeitreihenvisualisierungen. Zur Darstellung eines interaktiven Kartenelements wird Leaflet zusammen mit Karten des Anbieters CARTO (CartoDB, Inc., 2016) verwendet.

Die erfolgreiche Umsetzung des interaktiven Prototyps hat gezeigt, dass Shiny in Kombination mit ggplot2, Plotly und Leaflet als Framework für ein Visualisierungsprogramm geeignet ist. Aus diesem Grund fiel die Entscheidung, die Pakete in dieser Kombination einzusetzen.

6.4 Implementationsdetails

Dieser Abschnitt widmet sich den technischen Grundlagen der Applikation. Es wird dabei nicht die komplette Funktionsweise behandelt, sondern nur auf Implementationen eingegangen, welche einen erhöhten Erklärungsbedarf aufweisen.

6.4.1 Datenmodell Visualisierungstool

Für den Einsatz in Visualisierungen müssen die Daten relational, also tabellarisch vorliegen. Aus diesem Grund werden die Objekte aus der Datenbank beim Laden in Tabellenform zwischengespeichert. Im Backend unterscheidet das Tool zwischen „rawdata“, „data“ und „calculatedData“:

- rawdata: Direkt aus der Datenbank geladene Daten ohne Filter oder Datenmanipulation
- calculatedData: Berechnete Datensätze, nur temporär vorhanden
- data: Rawdata mit angewendeten Filtern und Datenmanipulation vereinigt mit calculatedData

Durch die Trennung dieser drei Kategorien wird sichergestellt, dass die Daten nicht verwechselt werden. Wichtig ist, dass data von rawdata und calculatedData abhängt und somit reaktiv aktualisiert wird, wenn sich diese beiden Variablen ändern.

Jede der drei „data“ Variablen ist identisch aufgebaut und besteht aus folgenden Tabellen:

- Datensätze / Datasets (ds)
- Eigenschaften / Properties (prop)
- Zeitreihendaten / TemporalData (td)
- Geographische Daten / SpatialData (sd)
- Eventbasierte Daten / EventData (ed)
- Lokalisierungsinformationen (loc)

Die Tabellen werden über IDs verknüpft und enthalten alle Eigenschaften der Datenbankobjekte. Tabelle 6.1 zeigt die TemporalData Tabelle. Die Abkürzung „td“ bedeutet TemporalData, steht also für Zeitreihenobjekte. „ds“ steht für Dataset, verweist also auf den Datensatz, zu dem der Wert gehört.

ds.ID	td.ID	td.datetime	td.value	td.unit
815	56646	1474919958	156.54	"MWh"

Tabelle 6.1: Beispiel der TemporalData (Zeitreihendaten) Tabelle von rawdata

Neben den drei „data“ Variablen gibt es fünf weitere, reaktive Objekte:

- `reac`: Dieses Objekt enthält temporäre Variablen, welche für die Graph-Visualisierungen benötigt werden.
- `mapObjects`: Dies enthält die Objekte, welche auf der Karte dargestellt werden (Punkte, Linien und Polygone von geographischen Daten und Lokalisierungen).
- `settings`: Hier werden Einstellungen gespeichert, welche beim Starten einer Benutzersession definiert werden.
- `datefilter`: Dient als Zwischenspeicher der beiden zeitlichen Filter.
- `leafletMap`: Dient als Zwischenspeicher für das Kartenvisualisierungsobjekt, welches von Leaflet erzeugt wurde.

6.4.2 Data Overview Visualisierung

Die Visualisierung im Tab Data Overview zeigt eine Übersicht über die in der Datenbank befindlichen Datensätze. Diese müssen also jedes Mal neu aus der Datenbank gelesen werden, um stets eine aktuelle Ansicht zu erzeugen. Hierfür werden in einem ersten Schritt die Namen aller Datensets abgefragt. Eine zweite Query sucht nach Supersets, um die Verbindungen zu den darin enthaltenen Datensätzen anzeigen zu können. Die Namen und Verbindungen werden als „nodes“ und „edges“ an das Paket „visNetwork“ weitergegeben, welches daraus eine Graph-Visualisierung zeichnet.

6.4.3 Daten laden

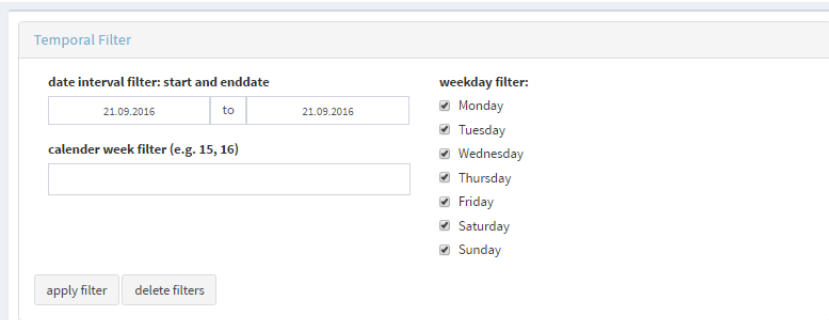
Klickt der Benutzer im Data Overview auf einen Datensatz, wird dieser aus der Datenbank in die rawdata Tabellen geladen. Um dem Benutzer eine Zusammenfassung der geladenen Daten anzuzeigen, werden zuerst die Eigenschaften ausgelesen. Es werden die Anzahl Zeitreihendaten, Anzahl Geodaten und Anzahl Event-Daten angezeigt sowie ob das Datenset eine Lokalisierung aufweist. Die Informationen sind in einer Tabelle unterhalb der Graph-Visualisierung zu finden. Im Anschluss werden die restlichen Tabellen über separate Queries befüllt.

6.4.4 Filtern

Im Tab Data Overview stehen dem Benutzer zwei Filtermöglichkeiten zur Verfügung. Die erste ist ein Textfilter, mit welchem die im Graph dargestellten Datensätze durchsucht werden können. Erfasst der Benutzer einen Filtertext, wird keine neue Datenbankabfrage ausgeführt. Stattdessen werden die vorher abgefragten Datensatznamen gefiltert. Im Anschluss wird der Graph aktualisiert.

Als zweiter Filter steht ein zeitlicher Filter bereit (siehe Abbildung 6.8). Dieser hat drei Funktionen: Er kann nach Start- und Enddatum, nach Kalenderwochen und nach Wochentagen filtern. Im Tab Data Overview werden sowohl Kalenderwochen, als auch Wochentage über die TimeObjects gefiltert, wobei eine neue Query ausgelöst wird. Der Graph zeigt also nur noch diejenigen Datensätze an, welche Informationen zu den gewählten Zeiträumen enthalten.

Der zeitliche Filter ist sowohl im Tab Data Overview, als auch im Tab Visualization verfügbar. Leider konnte nicht das gleiche Element benutzt werden, da jedes HTML Element eine einzigartige ID besitzen muss, welche Shiny nur pro Element erzeugt. Stattdessen wird eine Kopie mit geänderten IDs angezeigt. Änderungen im einen Filter werden im anderen stets übernommen, sodass sich dies nicht auf die Benutzerfreundlichkeit auswirkt. Allerdings muss dieser Umstand beachtet werden, wenn die Applikation gewartet wird. Änderungen müssen in diesem Fall jeweils an beiden Filtern vorgenommen werden.



Temporal Filter

date interval filter: start and enddate

21.09.2016 to 21.09.2016

calender week filter (e.g. 15, 16)

apply filter delete filters

weekday filter:

- ☒ Monday
- ☒ Tuesday
- ☒ Wednesday
- ☒ Thursday
- ☒ Friday
- ☒ Saturday
- ☒ Sunday

Abbildung 6.8: zeitliche Filtermöglichkeiten

6.4.5 Data Exploration

Der Data Exploration Tab zeigt Datensätze, welche den bereits geladenen Datensätzen ähnlich sind. In der Applikation sind 5 Methoden definiert, nach denen nach ähnlichen Daten gesucht wird:

- Gleiche Datenquelle: Zeigt alle Datensätze mit derselben Datenquelle
- Gleiche Zeitreihenart: Zeigt alle Datensätze, welche denselben Messabstand zwischen Zeitwerten besitzen (bspw. 15-Minütlich)
- Gleiche Zeitspanne: Zeigt alle Datensätze, welche mit den gleichen Time-Objects verknüpft sind (also denselben Zeitbereich behandeln)
- Gleiche Lokalisierung: Zeigt alle Datensätze, welche Daten über das gleiche Geo-Objekt enthalten
- Super- und Subsets anzeigen: Zeigt alle Supersets und alle Datensätze, welche in diesen Supersets enthalten sind sowie alle Subsets der geladenen Daten

Die Datensätze werden in drei Variablen geladen (siehe Programmcode 6.2). „originals“ enthält alle geladenen Datensätze. Diese werden mit dem Verbindungsobjekt aus „connectionNodes“ verbunden (beispielsweise an die Datenquelle, wenn die erste Methode aktiviert ist). An das Suchobjekt werden dann alle ähnlichen Datensätze „relatedDatasets“ gehängt.

Um den interaktiven Graph übersichtlich zu halten, sind die Verbindungen mit der Suchmethode beschriftet (siehe Abbildung 6.9). Der Graph wird zudem in Levels (Stufen) aufgebaut, sodass geladene Datensets stets auf der linken Seite und ähnliche Datensets stets auf der rechten Seite zu finden sind. Dazwischen werden die Gemeinsamkeiten angezeigt.

```
# changed for better comprehension from Server.R:557
# get datasets with similar datasources

query <- paste0("MATCH (n:Dataset) ",
               "WHERE ID(n) IN [", paste(rawdata$ds[[1]], collapse = ", "), "]",
               "RETURN ID(n) as ID, n.name as name, n.description as description, 'original' as type")
originals <- cypher(graph, query)

if ("datasource" %in% settings$exploration_types) {
  query <- paste0("MATCH (n:Dataset)-[:SOURCED_FROM]->(src:Datasource)-[:SOURCED_FROM]-(rel:Dataset) ",
               "WHERE ID(n) IN [", paste(rawdata$ds[[1]], collapse = ", "), "]",
               "RETURN ID(n) as IDsfrom, ID(src) as IDconnection, ID(rel) as ID, rel.name as name, ",
               "rel.description as description, 'datasource' as type"
  )
  relatedDatasets <- rbind(relatedDatasets, cypher(graph, query))

  query <- paste0("MATCH (n:Dataset)-[:SOURCED_FROM]->(src:Datasource) ",
               "WHERE ID(n) IN [", paste(rawdata$ds[[1]], collapse = ", "), "]",
               "RETURN ID(src) as ID, src.name as name, 'datasource' as type"
  )
  connectionNodes <- rbind(connectionNodes, cypher(graph, query))
}
```

Programmcode 6.2: Codeausschnitt für das Laden von Datensets mit gleicher Datenquelle

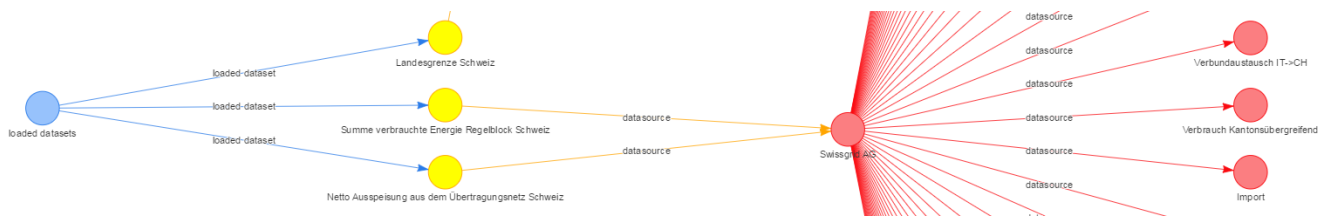


Abbildung 6.9: abgestufte Darstellung im Tab Data Explorer

6.4.6 Dynamisches UI-Element

Das „Show / Hide“ Element im Tab Visualization wird anhand der geladenen Datensets aufgebaut und erlaubt es, Teile davon auszublenden. Da die geladenen Datensets von Benutzereingaben abhängen, muss dieses Element dynamisch erstellt werden. Für diesen Zweck bietet Shiny „renderUI“ Funktionen, mit denen serverseitig UI-Code generiert werden kann.

Das „Show / Hide“ Element enthält Benutzereingabe-Felder. Auch deren IDs müssen in der Applikation einzigartig sein. Aus diesem Grund werden diese IDs dynamisch auf Basis der Datensatz-ID generiert. Die Inputvariablen können dann im Programmablauf benutzt werden, indem das von Shiny bereitgestellte „input“-Objekt auf eine spezielle Art ausgelesen wird (siehe Programmcode 6.3).

```
# Normales Auslesen
myVar <- input$myVar

# Dynamisches Auslesen
name <- "myVar"
myVar <- input[[name]]
```

Programmcode 6.3: Vergleich normales / dynamisches Auslesen des input-Objekts

6.4.7: Visualisierungscode

Die Zeitreihen- und Kartenvisualisierungen werden anhand eines pro Benutzersession definierten Standardcode generiert. Will der Benutzer die Visualisierungen anpassen, kann er direkt den ausgeführten Code ändern. Er verändert also die in der Benutzersession abgelegten Variablen „plotCode“ (siehe Programmcode 6.4) und „mapCode“ (siehe Programmcode 6.5). Mit Klick auf den „apply code“ Button wird dieser Code über eine „eval“-Funktion in R Code umgewandelt und ausgeführt. Auf diese Weise kann der gesamte Funktionsumfang von ggplot2, Plotly und Leaflet verwendet werden, welche alle Visualisierungsanforderungen abdeckt.


```

gg <- ggplot()
# if temporaldata is available
if (length(data$td) > 0) {
  # go through all datasets
  td.ds <- distinct(data$td)[1][1]
  for (i in 1:length(td.ds)) {
    # get temporal data of dataset
    line.data <- filter(data$td, IDds == td.ds[i])
    # add td as line to gg
    gg <- gg + geom_line(aes(td.datetime, td.value),
                        alpha = 0.5, colour = settings$pal(td.ds[i]),
                        data = line.data)
  }
}
# convert ggplot2 object into interactive plotly object
ggplotly(gg)

```

Programmcode 6.4: Standardcode für Zeitreihenvisualisierung

```

# Create leaflet map with CartoDB.Positron background images
map <- leaflet() %>% addProviderTiles('CartoDB.Positron')
# add polygon geoobjects of loaded datasets
if (!is.null(mapObjects$polygons)) {
  map <- map %>%
    addPolygons(data = mapObjects$polygons, color = ~settings$pal(IDds),
    fillColor = ~settings$pal(IDds))
}
# add localization polygons
if (!is.null(mapObjects$loc_polygons)) {
  map <- map %>%
    addPolygons(data = mapObjects$loc_polygons, color = ~set-
    tings$pal(IDds), fillColor = ~settings$pal(IDds))
}
# return map
map

```

Programmcode 6.5: Standardcode für Kartendarstellung

6.4.8 Export

Zeitreihenvisualisierungen, welche über Plotly dargestellt werden, können direkt in der Visualisierung als Bilddatei exportiert werden. Dies ist bereits im Funktionsumfang von Plotly integriert.

Das für die Kartenvisualisierung benutzte Paket Leaflet bringt keine solche Funktion mit, weshalb diese manuell implementiert werden musste (siehe Programmcode 6.6). Neben der Kartenvisualisierung wurde ein Button für den Download eingefügt. Wird darauf geklickt, generiert das Paket „htmlwidgets“ eine temporäre HTML Datei mit der Kartendarstellung. Leaflet rendert Karten aber erst, wenn sie aufgerufen werden. Die HTML-Datei muss also mit einem separaten Paket „webshot“ in eine sichtbare Karte umgewandelt werden. Dazu wird das Paket „PhantomJS“ benutzt, eine Javascript-API welche vor der ersten Benutzung von webshot installiert werden muss. Es öffnet die HTML-Datei in einem Browser-Tab und überspeichert die temporäre Datei mit der darin nun geladenen Webseite. Die sichtbare Karte wird dem Browser als PDF-Datei zum Download übergeben.

```

# Download map visualization
output$mapVizDownload <- downloadHandler(
  filename = 'plot.pdf',

  content = function(file) {
    # temporarily switch to the temp dir, in case you do not have write
    # permission to the current working directory
    owd <- setwd(tempdir())
    on.exit(setwd(owd))

    saveWidget(leafletMap(), "temp.html", selfcontained = FALSE)
    webshot("temp.html", file = file, cliprect = "viewport")
  }
)

```

Programmcod 6.6: Export der Kartenvisualisierung

6.4.9 Gliederung des Codes

Der Server-Programmcod ist in fünf Sektionen eingeteilt, um die Lesbarkeit und die Übersicht zu verbessern. Dieser Aufbau vereinfacht ausserdem das Debugging (die Fehlersuche). Es muss nie rekonstruiert werden, zu welchem Zeitpunkt ein Codeabschnitt ausgeführt wird. Dies ist immer aus dem Kontext ersichtlich.

Die erste Sektion dient zum Aufsetzen der Arbeitsumgebung. Dieser Teil des Programms wird ausgeführt, wenn die Applikation startet. Es werden zuerst alle benötigten Pakete geladen und einige Parameter gesetzt. Im Anschluss wird das Neo4jHelper-Script geladen, welches erweiterte Funktionen zur Interaktion mit der Datenbank bereitstellt (siehe Abschnitt B.1 im Anhangs B). Ausserdem wird die Verbindung zur Datenbank aufgebaut.

Die zweite Sektion „Startup Code“ wird beim Starten einer neuen Benutzersession ausgeführt. Es wird jeweils eine Session pro Browser-Tab erstellt.

Im dritten Abschnitt werden alle reaktiven Variablen definiert. Diese können in darauffolgenden Code-Abschnitten als normale Variablen behandelt werden.

Die vierte Sektion definiert alle „Observer“-Funktionen. Ein Observer ist ein Konstrukt der reaktiven Programmierung. Alle im Code des Observers verwendeten reaktiven Variablen inklusive Benutzereingaben werden ständig beobachtet. Ändert sich eines der Objekte, wird der Code reaktiv neu ausgeführt. So können Kettenreaktionen programmiert werden, welche Benutzereingaben in mehreren Arbeitsschritten in eine neue Darstellung umwandeln.

Die fünfte Sektion definiert alle dynamischen Anzeigen, wie zum Beispiel die Zeitreihenvisualisierung oder die Kartendarstellung. Wie die Observer sind sie von reaktiven Variablen abhängig und werden bei Bedarf aktualisiert.

Auch innerhalb der Sektionen ist der Code gegliedert. Hier wird der Code nach Tabs sortiert, wobei folgende Reihenfolge angewendet wird:

- Code von Tab Database Overview
- Code von Tab Data Explorer
- Code von Tab Visualization
- Code von Tab Raw Data

6.4.10 Code-Kommentare

Da die Visualisierungsapplikation nach Abschluss der Bachelorarbeit weiterentwickelt werden soll, wurde von Anfang an viel Wert auf Codequalität gelegt. Ein wichtiger Punkt davon ist das Kommentieren von Funktionen und Abschnitten, was die Lesbarkeit stark erhöht. Im Programmcode des Visualisierungsprogramms sind alle Funktionen kommentiert. Innerhalb der Funktionen sind an den wichtigen Stellen zusätzliche Anmerkungen angebracht, welche die Funktionsweise nochmals genauer erklären.

Es gibt kein optimales Verhältnis von Kommentarzeilen zu Codezeilen. Dies ist stark abhängig davon, wie das Programm gepflegt und weiterentwickelt wird. In diesem Fall wird die Applikation zu Beginn vom Betreuer der Arbeit, Dr. Sinan Teske, gewartet. Um ihm die Arbeit zu erleichtern, wurde hier ein hoher Anteil an Kommentaren bevorzugt. So sind im Backend 19 % der Codezeilen Kommentare.

Kapitel 7

Diskussion der Resultate

In diesem Kapitel soll das erstellte Datenmodell und die implementierte Applikation kritisch angeschaut werden. Wie in den anderen Teilen der Arbeit wird das Datenmodell und die Visualisierungssapplikation getrennt betrachtet.

7.1 Eignung des Datenmodells

Das Ziel war ein Datenmodell zu entwickeln, welche die aktuell vorhandenen Datensätze in ein vergleichbares Format bringt. Überdies sollte es zukünftigen Anforderungen standhalten, also auch noch nicht berücksichtigte Daten aufnehmen können. Dieses Ziel stellt ein Grundziel dieser Bachelorarbeit dar, da bei Nichterreichen alle weiteren Ziele gefährdet wären. Aus diesem Grund wurde entsprechend viel Zeit investiert, um ein optimales Ergebnis zu erreichen.

Das vorliegende Datenmodell wurde durch ein iteratives Vorgehen erstellt und ständig verbessert. In Brainstorming-Sitzungen haben wir zukünftig verfügbare Daten gesammelt und getestet, ob das Datenmodell diese aufnehmen kann. Das Ergebnis ist positiv. Das Datenmodell wurde bewusst einfach gehalten, sodass verschiedenste Daten aus dem Energiesektor als einzelne Datensätze aufgenommen werden können. Mit dem Einfügen in das Datenmodell werden diese vergleichbar, wobei ein Vergleich nicht in jedem Fall sinnvoll ist. Nach der Implementierung des Visualisierungstools konnte das Datenmodell im Praxiseinsatz getestet werden, wo es durch performantes Laden und Bearbeiten der Daten überzeugte. Der von Neo4j bereitgestellte Neo4j-Browser konnte aber nicht vollständig die Anforderungen erfüllen, weshalb mit dem Neo4jHelper-Script eine zweite Möglichkeit entwickelt wurde, die Datenbank zu verwalten.

7.2 Eignung des Visualisierungstools

Ziel des Visualisierungstools ist es, die Arbeit von Wissenschaftlern im Energiebereich zu unterstützen. Das Tool wurde anhand von Anwendungsfällen konzipiert, welche in Gesprächen mit Dr. Sinan Teske ausgearbeitet wurden. Aus diesen wurde eine Reihe von Features abgeleitet.

Das Tool erfüllt alle kritischen Anforderungen. So ist es an die Datenbank angeschlossen und ermöglicht die Visualisierung von Datensätzen. Sowohl Zeitreihen als auch Kartendarstellung werden unterstützt und lassen sich in interaktiver Art

und Weise betrachten. Über Export-Möglichkeiten kann man die Visualisierungen als Grafiken in wissenschaftlichen Arbeiten Einbauen. Durch das Einbinden der Code Editoren können die Visualisierungen umfassend bearbeitet und nach eigenem Ermessen verändert werden. Zusätzlich sind die Grundlagen dazu geschaffen, berechnete Datensets zu erstellen. Hier fehlen nur Helferfunktionen, welche dies erleichtern. Auch ein Data Explorer ist vorhanden, welcher auf ähnliche Datensets hinweist. Durch die Implementierung mit R und Neo4j wurde die Anforderung, quelloffene Programme zu benutzen, vollständig erfüllt.

Da die Erarbeitung der Konzeption mehr Zeit als erwartet in Anspruch nahm, konnten nicht alle Anforderungen implementiert werden. Es mussten also Prioritäten gesetzt werden, nach denen die Features implementiert wurden. So wurde die Koppelung von Geodaten und Zeitreihen noch nicht implementiert. Hiermit wäre es möglich, lokalisierte Zeitreihen auf der Karte darzustellen und zu analysieren. Es ist ausserdem noch nicht möglich, mehrere Visualisierungen gleichzeitig zu erstellen oder den Arbeitsstand zwischenspeichern.

Trotz den fehlenden Funktionen ist das Visualisierungstool einsatzbereit. Durch die Datenbankbindung sind die Daten sehr schnell verfügbar. Durch wenige Klicks erhält der Benutzer eine Visualisierung, welche in wissenschaftlichen Arbeiten eingesetzt werden kann. Dies ist ein grosser Fortschritt gegenüber herkömmlichen Methoden. Dort mussten die Datensätze jeweils separat eingelesen, in eine vergleichbare Form gebracht und anschliessend visuell aufbereitet werden. Das Visualisierungstool nimmt dem Benutzer die Vorbereitungsarbeit ab, womit er sich auf die Erstellung aussagekräftiger Visualisierungen konzentrieren kann. Über den Data Explorer erlaubt das Tool eine Kontrolle der eingesetzten Daten. In der herkömmlichen Vorgehensweise sind diese in verschiedenen Ordnern auf der Festplatte hinterlegt, somit kann schnell ein Datensatz vergessen werden. Mit dem Data Explorer werden einem alle ähnlichen Daten aufgezeigt, sodass diese Datensätze gefunden werden können.

8.3 Ausblick

Im Moment wird die Datenbank lokal auf einem Computer gespeichert und ausgeführt. Um diese produktiv in Betrieb zu nehmen, soll sie auf einen Server umgezogen werden. Im Anschluss wird sie mit Daten befüllt werden, wobei die Beispieldatensätze gelöscht werden.

Die vorliegende Fassung des Visualisierungstools soll nicht die Endfassung darstellen, da nicht alle Anforderungen des Konzeptes vollständig umgesetzt werden konnten. Dies soll nun im Anschluss an diese Arbeit geschehen. Die fehlenden Funktionalitäten sind zwar unkritisch, würden den Benutzern aber helfen, produktiver arbeiten zu können. Der momentane Stand stellt eine gute Basis für die

Weiterentwicklung dar. Durch das entworfene Konzept war schon von Anfang klar, welche Funktionalitäten die Endfassung haben sollte. Die Codebasis der Applikation ist also darauf vorbereitet, um diese Funktionen erweitert zu werden. Der Arbeitsaufwand, um diese Verbesserungen durchzuführen, liegt bei etwa zwei Monaten. Eine Aufführung aller noch nicht umgesetzten Elemente der Konzeption ist als Anhang C angefügt.

Quellenverzeichnis

- Becher, R. (2016). *ralfbecher/tableau-neo4j-wdc*. Abgerufen am 11. September 2016 von GitHub: <https://github.com/ralfbecher/tableau-neo4j-wdc>
- Bonér et al., J. (2014). Abgerufen am 27. September 2016 von Das Reaktive Manifest: <http://www.reactivemanifesto.org/de>
- Bridge, R. (3. November 2016). *Open Source Software – The Advantages & Disadvantages*. Von Entrepreneur Handbook: <http://entrepreneurhandbook.co.uk/open-source-software/> abgerufen
- Bundesamt für Energie. (2016). *Energiestrategie 2050*. Abgerufen am 28. September 2016 von Bundesamt für Energie BFE: http://www.bfe.admin.ch/energiestrategie2050/index.html?lang=de&dossier_id=06543
- CartoDB, Inc. (2016). *Basemaps Data Services*. Abgerufen am 18. September 2016 von CARTO: <https://carto.com/location-data-services/basemaps/>
- DB-Engines Ranking*. (2016). Abgerufen am 9. September 2016 von DB-Engines: <http://db-engines.com/de/ranking>
- de Vries, A. (2016). *On the growth of CRAN packages*. Abgerufen am 19. September 2016 von R-Bloggers: <https://www.r-bloggers.com/on-the-growth-of-cran-packages/>
- deldersveld. (2016). *Power BI Community*. Abgerufen am 13. September 2016 von Enhanced Map with Custom Geography - Best Visuals Contest: <https://community.powerbi.com/t5/user/viewprofilepage/user-id/170>
- Diakopoulos, N., & Cass, S. (2016). *Interactive: The Top Programming Languages 2016*. Abgerufen am 19. September 2016 von IEEE Spectrum: Diakopoulos
- Duden. (2016). *Graph*. Abgerufen am 22. September 2016 von Duden: http://www.duden.de/rechtschreibung/Graph_Darstellung_Mathematik
- ElCom. (2016). *Berichte und Studien*. Abgerufen am 29. September 2016 von Eidgenössische Elektrizitätskommission ElCom: <https://www.elcom.admin.ch/elcom/de/home/dokumentation/berichte-und-studien.html>
- ESRI, Inc. (2016). *Shapefiles*. Abgerufen am 28. September 2016 von ESRI: <https://doc.arcgis.com/de/arcgis-online/reference/shapefiles.htm>

- Garling, C. (2012). *Amazon Goes Back to the Future With 'NoSQL' Database*. Abgerufen am 22. September 2016 von Wired: <https://www.wired.com/2012/01/amazon-dynamodb/>
- Geissmann, T. (2016). *Zur Wirtschaftlichkeit der Wasserkraft*. Abgerufen am 28. September 2016 von ETH Zürich: <https://www.ethz.ch/de/news-und-veranstaltungen/eth-news/news/2016/04/zur-wirtschaftlichkeit-der-wasserkraft.html>
- GitHub. (2016). *neo4j-contrib/spatial*. Abgerufen am 10. September 2016 von GitHub: <https://github.com/neo4j-contrib/spatial>
- Graphendatenbank. (2013). Abgerufen am 11. September 2016 von edb-Online-Datenbanklexikon: http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Graphendatenbank
- Grolemund, G. (2015). *How to understand reactivity in R*. Abgerufen am 26. September 2016 von Rstudio: <http://shiny.rstudio.com/articles/understanding-reactivity.html>
- Gull, C. (2012). *Web-Applikationen entwickeln mit NoSQL*. Franzis Verlag.
- Hadley, W. (2013). *ggplot2*. Abgerufen am 5. September 2016 von ggplot2: <http://ggplot2.org>
- Holzgraefe, H. (2016). *GIS features in MariaDB and MySQL*. Abgerufen am 19. September 2016 von FrOSCon: https://programm.froscon.de/2016/system/event_attachments/attachments/000/000/388/original/MariaDB_MySQL_GIS_Features.pdf
- Iseminger, D. (2016a). *Power BI Documentation*. Abgerufen am 13. September 2016 von Data sources in Power BI Desktop: <https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-data-sources/>
- Iseminger, D. (2016b). *DAX basics in Power BI Desktop*. Abgerufen am 14. September 2016 von Power BI Documentation: <https://powerbi.microsoft.com/en-us/documentation/powerbi-desktop-quickstart-learn-dax-basics/>
- Leaflet for R. (2015). *Introduction*. Abgerufen am 5. September 2016 von Leaflet for R: <https://rstudio.github.io/leaflet/>
- Microsoft. (2016a). *Microsoft*. Abgerufen am 13. September 2016 von Power BI: <https://powerbi.microsoft.com/de-de/features/>
- Microsoft. (2016b). *Power BI*. Abgerufen am 13. September 2016 von Custom Visuals Gallery: Microsoft

- Muehsig, R. (2009). *HowTo: Multithreading in .NET - Asynchrone Programmierung (Eventbasiert)*. Abgerufen am 26. September 2016 von Code Inside Blog: <http://blog.codeinside.eu/2009/04/12/howto-multithreading-in-net-asynchrone-programmierung-eventbasiert/>
- MySQL Reference Book*. (2016). Abgerufen am 5. September 2016 von MySQL: <https://dev.mysql.com/doc/refman/5.7/en/spatial-analysis-functions.html>
- Neo Technology, Inc. (2016a). *Neo4j Open Source Project*. Abgerufen am 10. September 2016 von Neo4j: <https://neo4j.com/open-source-project/>
- Neo Technology, Inc. (2016b). *Neo4j Java Developer Reference v3.0*. Abgerufen am 14. September 2016 von Neo4j: <https://neo4j.com/docs/java-reference/current/#server-extending>
- Neo Technology, Inc. (2016c). *Our Customers*. Abgerufen am 19. September 2016 von Neo4j: <https://neo4j.com/customers/>
- Neo Technology, Inc. (2016d). *Graph Data Modeling Guidelines*. Abgerufen am 18. September 2016 von Neo4j: <https://neo4j.com/developer/guide-data-modeling/>
- Neo Technology, Inc. (2016e). *Intro to Cypher*. Abgerufen am 21. September 2016 von Neo4j: <https://neo4j.com/developer/cypher-query-language/>
- Neo Technology, Inc. (2016f). *Chapter 3. Cypher query language*. Abgerufen am 22. September 2016 von The Neo4j Developer Manual v3.0: <https://neo4j.com/docs/developer-manual/current/cypher/>
- NOSQL Databases*. (2016). Abgerufen am 11. September 2016 von <http://nosql-database.org>
- plotly. (2016). *Plotly R Library 2.0*. Abgerufen am 5. September 2016 von plotly: <https://plot.ly/r/>
- PostGIS Features*. (2016). Abgerufen am 10. September 2016 von PostGIS: <http://postgis.net/features/>
- Qlik. (2013). *Personal Edition FAQs*. Abgerufen am 19. September 2016 von Qlik Community: <https://community.qlik.com/docs/DOC-1799>
- Qlik. (2016a). *QlikView Produktübersicht*. Abgerufen am 13. September 2016 von Qlik: <http://www.qlik.com/de-de/products/qlikview>
- Qlik. (2016b). *QlikView Qlik Map*. Abgerufen am 14. September 2016 von Qlik Market: <http://market.qlik.com/qlikview-qlik-map.html>
- R Foundation. (2016). *The R Project for Statistical Computing*. Abgerufen am 19. September 2016 von The R Project: <https://www.r-project.org>

- r-bloggers. (2016). Abgerufen am 19. September 2016 von r-bloggers:
<https://www.r-bloggers.com>
- RStudio. (2016). *Put Shiny Web Apps Online*. Abgerufen am 5. September 2016 von RStudio Shiny Server: <https://www.rstudio.com/products/shiny/shiny-server/>
- Santos, P., Moro, M., & Davis, C. (2015). Comparative Performance Evaluation of Relational and NoSQL Databases for Spatial and Mobile Applications. *International Conference on Database and Expert Systems Applications*.
- Saxton, A. (2015). *Microsoft Power BI Blog*. Abgerufen am 16. September 2016 von The Conceptual Data Model and Limits:
<https://powerbi.microsoft.com/en-us/blog/the-conceptual-data-model-and-limits/>
- Schwaber, K., & Sutherland, J. (2013). *Der Scrum Guide*. Abgerufen am 10. September 2016 von SCRUM Guides:
www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf
- spaltenorientierte Datenbanksysteme*. (2014). Abgerufen am 11. September 2016 von edb-Datenbanken-Online-Lexikon: http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/SpaltenorientierteDatenbank
- Tableau Community. (2014). *Tableau Community*. Abgerufen am 13. September 2016 von Support for geometry (polygon) data types:
<https://community.tableau.com/ideas/3238>
- Tableau Software. (2016a). *Tableau*. Abgerufen am 13. September 2016 von Tableau Desktop: <https://www.tableau.com/de-de/products/desktop>
- Tableau Software. (2016b). *Tableau*. Abgerufen am 13. September 2016 von Datenvisualisierung: <https://www.tableau.com/de-de/stories/topic/data-visualization>
- Tableau Software. (2016c). *Tableau Reader*. Abgerufen am 13. September 2016 von Tableau: <http://www.tableau.com/de-de/upgrade-reader>
- Tagesanzeiger. (7. März 2016). *Schweizer Stauseen im Ausverkauf*. Abgerufen am 28. September 2016 von Tagesanzeiger:
<http://www.tagesanzeiger.ch/wirtschaft/standard/Alpiq-will-Haelfte-seiner-Stauseen-verkaufen/story/12729839>
- TIQ Solutions. (2016). *JDBC Connector for QlikView*. Abgerufen am 13. September 2016 von Qlik Market: <http://market.qlik.com/qlikview-jdbc-connector.html>
- Trello, Inc. (2016). Abgerufen am 19. September 2016 von Trello:
<https://trello.com>

White, N. (2016). *nicolewhite/RNeo4j*. Abgerufen am 19. September 2016 von
GitHub: <https://github.com/nicolewhite/RNeo4j>

Abbildungsverzeichnis

Abbildung 2.1: Anzahl verfügbarer Pakete pro R Version (de Vries, 2016).	5
Abbildung 2.2: reaktiver Programmablauf (verändert nach (Grolemund, 2015))..	6
Abbildung 5.1: Entwurf Eigenentwicklung	16
Abbildung 6.1: Trello Board eines Sprints	18
Abbildung 6.2: Datenmodell Graphdatenbank	19
Abbildung 6.3: TimeTree Baum mit Beispieldatensätzen.....	21
Abbildung 6.4: Beispiel einer Zeitreihe des Vorbilds.....	23
Abbildung 6.5: Prototyp interaktive Shiny Applikation.....	24
Abbildung 6.6: Übersicht der Tabs des Visualisierungstools	24
Abbildung 6.7: Struktureller Aufbau der Applikation mit eingesetzten Technologien	25
Abbildung 6.8: zeitliche Filtermöglichkeiten	29
Abbildung 6.9: abgestufte Darstellung im Tab Data Explorer	31
Abbildung A.1: Tab Data Overview mit geladenen Datensätzen und aufgeklapptem zeitlichem Filtermenü	48
Abbildung A.2: Tab Data Exploration mit zwei eingestellten Explorationstypen	49
Abbildung A.3: Tab Visualization mit einem geladenen Datenset, welches eine Lokalisierung aufweist.....	50
Abbildung A.4: Tab Raw Data, wenn keine Datensätze geladen sind.....	51
Abbildung A.5: Ausschnitt Tab Raw Data bei einem geladenen Datensatz.....	51
Abbildung B.1: Neo4j Browser mit Abfragebeispiel (Ergebnisse als Graph)	52
Abbildung B.2: TimeTree mit Beispieldatensätzen	54
Abbildung B.3: Beispiel eines angewendeten Textfilters	57
Abbildung B.4: ausgefüllter zeitlicher Filter im ausklappbaren Panel.....	58
Abbildung B.5: Übersicht geladene Daten mit zwei zum Löschen markierten Datensätzen	58
Abbildung B.6 Auswahl der Methoden um ähnliche Datensets zu finden	59
Abbildung B.7 Data Exploring mit zwei aktivierten Suchmethoden.....	60

Abbildung B.8: aufklappbare Panels, in welchem Datensetteile ausgeblendet werden können.....	61
Abbildung B.9: Code Editor, mit welchem berechnete Datensets erzeugt werden können	62

Tabellenverzeichnis

Tabelle 2.1: wichtigste Kategorien von NoSQL.....	4
Tabelle 4.1: Subjektive Bewertung der Erfüllung der Anforderungen.....	12
Tabelle 5.1: Subjektive Bewertung der Erfüllung der Anforderungen.....	17
Tabelle 6.1: Beispiel der TemporalData (Zeitreihendaten) Tabelle von rawdata	27

Programmcodeverzeichnis

Programmcode 6.1: Ausschnitt aus dem Script, welches Beispieldaten importiert	22
Programmcode 6.2: Codeausschnitt für das Laden von Datensets mit gleicher Datenquelle	30
Programmcode 6.3: Vergleich normales / dynamisches Auslesen des input-Objekts	31
Programmcode 6.4: Standardcode für Zeitreihenvisualisierung.....	32
Programmcode 6.5: Standardcode für Kartendarstellung.....	32
Programmcode 6.6: Export der Kartenvisualisierung	33
Programmcode B.1: Beispiel eines Zeitreihenimports	54
Programmcode B.2: Beispiel eines Imports von 2 Shapefiles.....	56
Programmcode B.3: Cypher Abfrage um Objekte im Neo4j Browser zu kontrollieren.....	56
Programmcode B.4: Cypher Abfrage um Objekte im Neo4j Browser zu kontrollieren.....	56
Programmcode B.5: Standardcode für Zeitreihenvisualisierung	62
Programmcode B.6: Standardcode für Kartendarstellung	63

Anhang A

Screenshots der Webapplikation

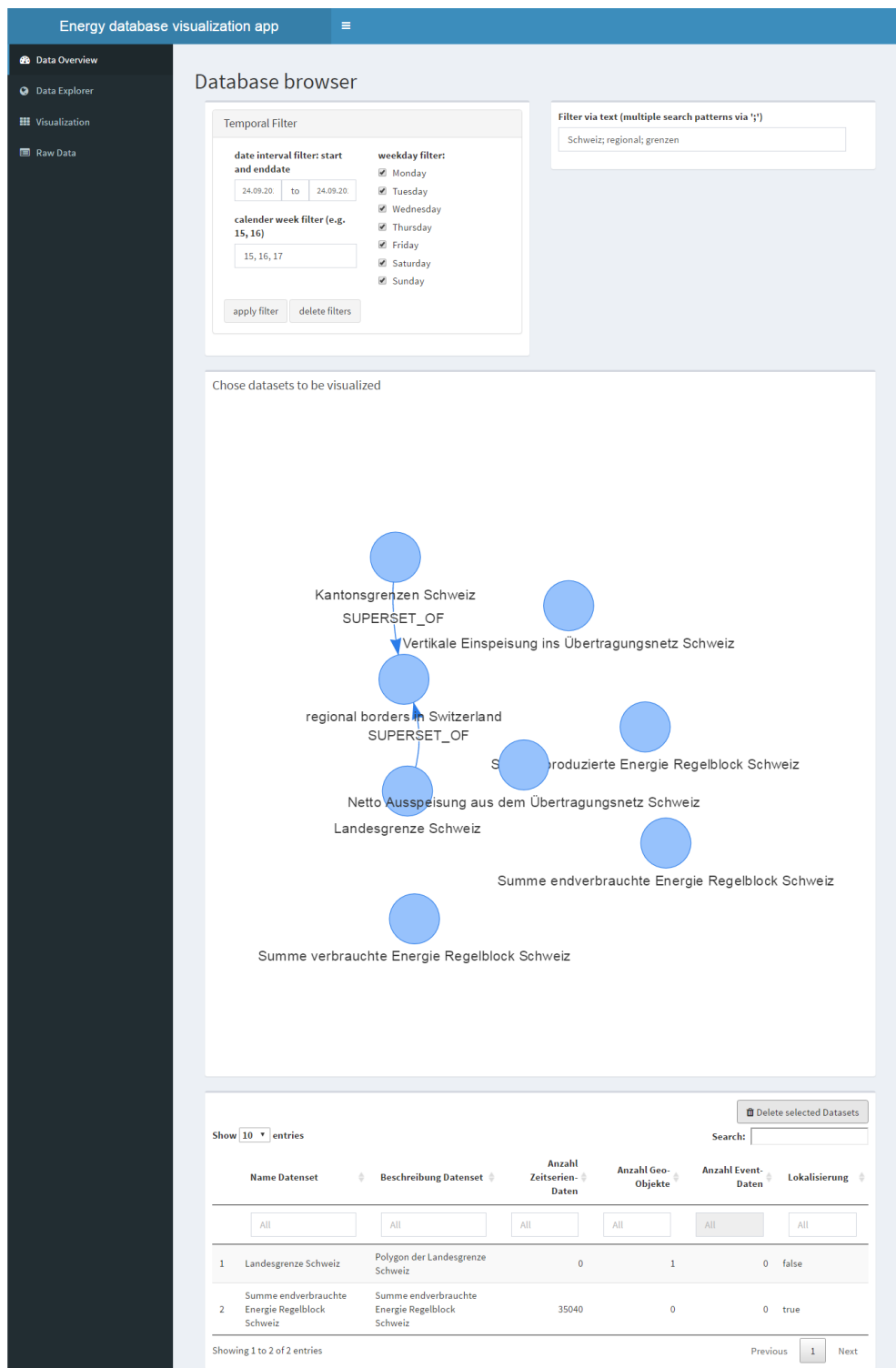


Abbildung A.1: Tab Data Overview mit geladenen Datensätzen und aufgeklapptem zeitlichem Filtermenü

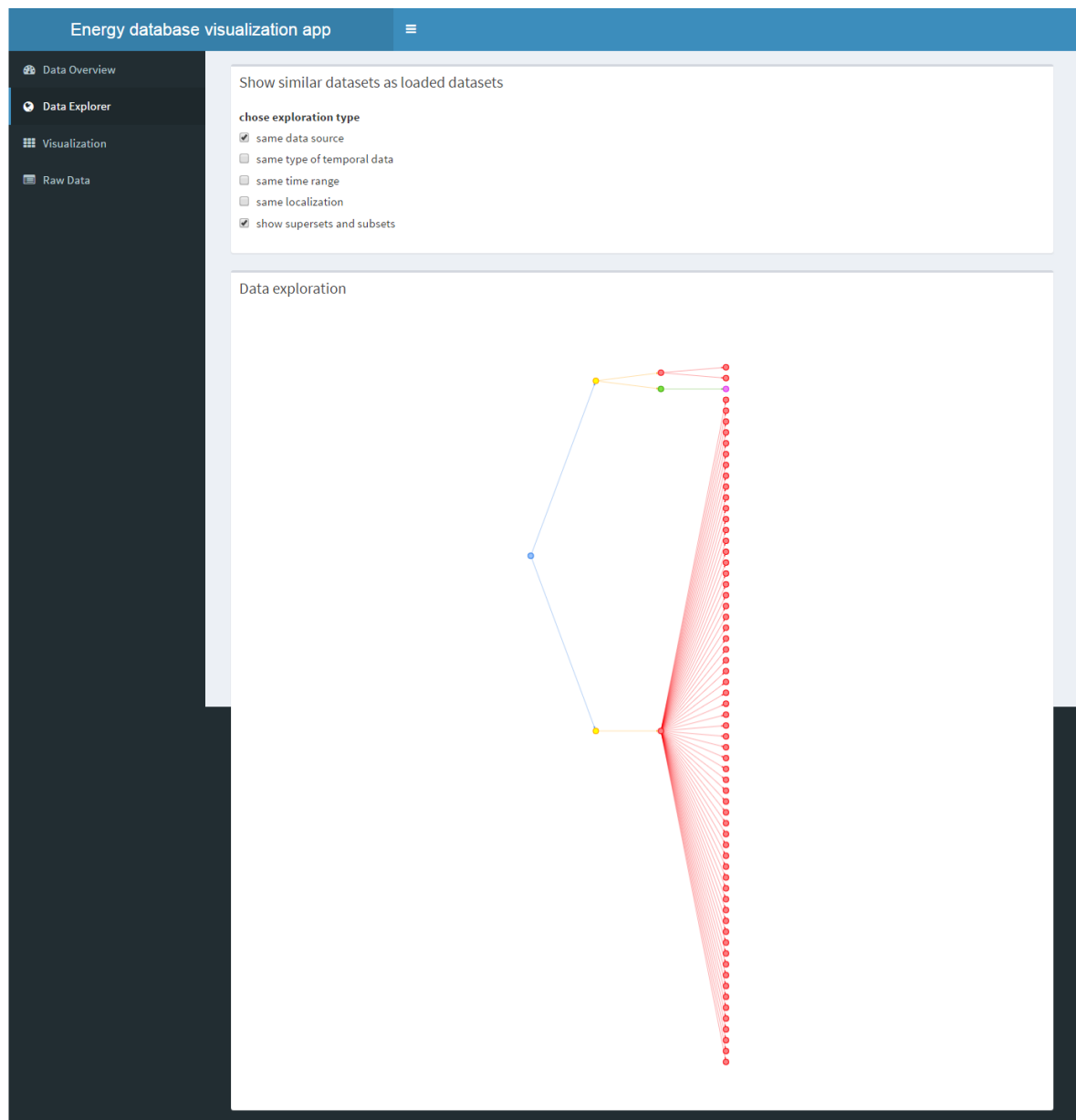


Abbildung A.2: Tab Data Exploration mit zwei eingestellten Explorationstypen

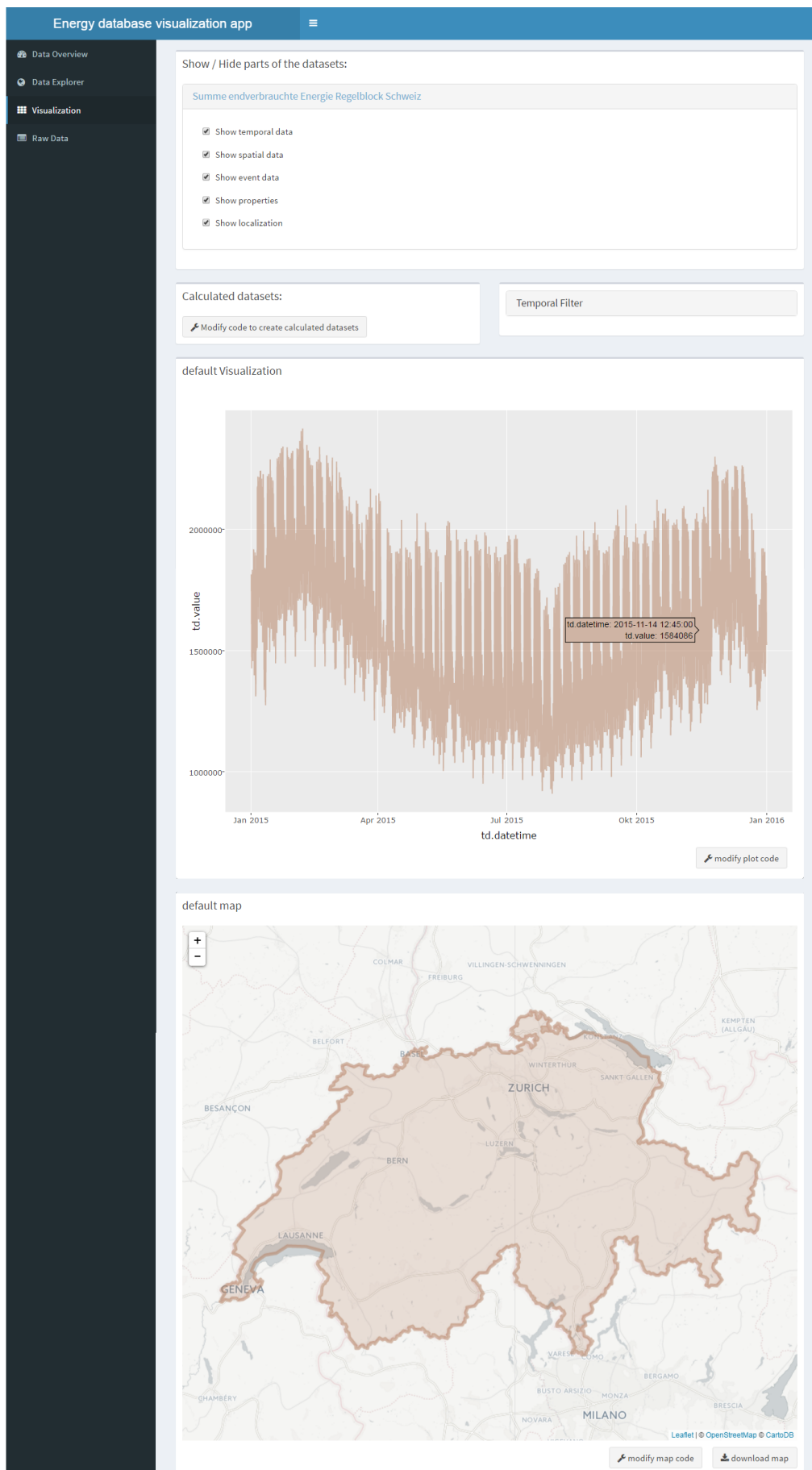


Abbildung A.3: Tab Visualization mit einem geladenen Datenset, welches eine Lokalisierung aufweist

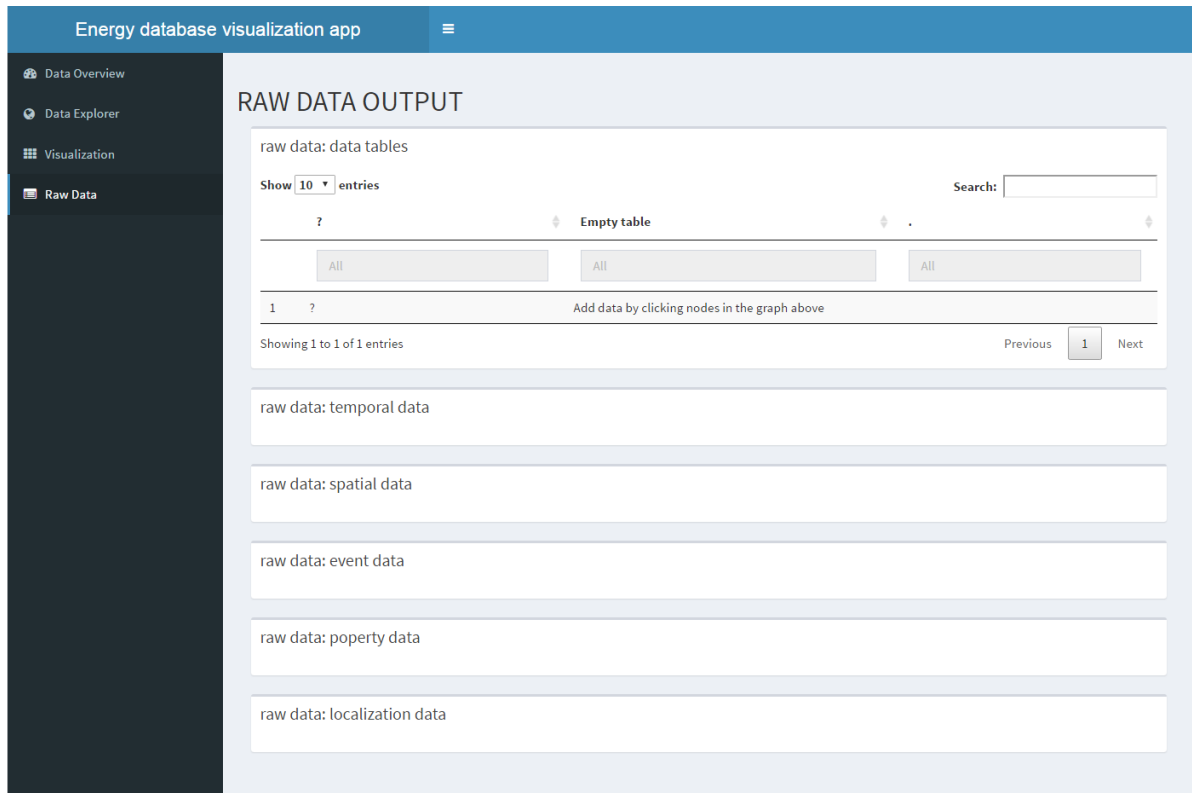


Abbildung A.4: Tab Raw Data, wenn keine Datensätze geladen sind

Energy database visualization app

Data Overview
Data Explorer
Visualization
Raw Data

RAW DATA OUTPUT

raw data: data tables

Show 10 entries Search:

	ID	Name Datenset	Beschreibung Datenset	Anzahl Zeitserien-Daten	Anzahl Geo-Objekte	Anzahl Event-Daten	Lokalisierung
1	1558	Summe endverbrauchte Energie Regelblock Schweiz	Summe endverbrauchte Energie Regelblock Schweiz	35040	0	0	true

Showing 1 to 1 of 1 entries Previous 1 Next

raw data: temporal data

Show 10 entries Search:

	IDds	IDtd	td.datetime	td.value	td.unit
1	1558	35122	2015-12-16T14:00:00Z	2091198.119	kWh
2	1558	35121	2015-12-16T13:45:00Z	2089741.193	kWh
3	1558	35120	2015-12-16T13:30:00Z	2100703.068	kWh
4	1558	35119	2015-12-16T13:15:00Z	2113729.951	kWh

Abbildung A.5: Ausschnitt Tab Raw Data bei einem geladenen Datensatz

Anhang B

Benutzerhandbuch

Visualisierungsapplikation

B.1 Datenbankmanagement

Für die Datenbankverwaltung stellt Neo4j eine Weboberfläche zur Verfügung, welche sich „Neo4j Browser“ nennt (siehe Abbildung B.1). Über eine Abfragensprache, welche sich bei Neo4j „Cypher“ (Neo Technology, Inc., 2016e) nennt, kann man die Datenbank durchsuchen. Die Ergebnisse werden je nach Query in Graph- oder Tabellenform angezeigt.

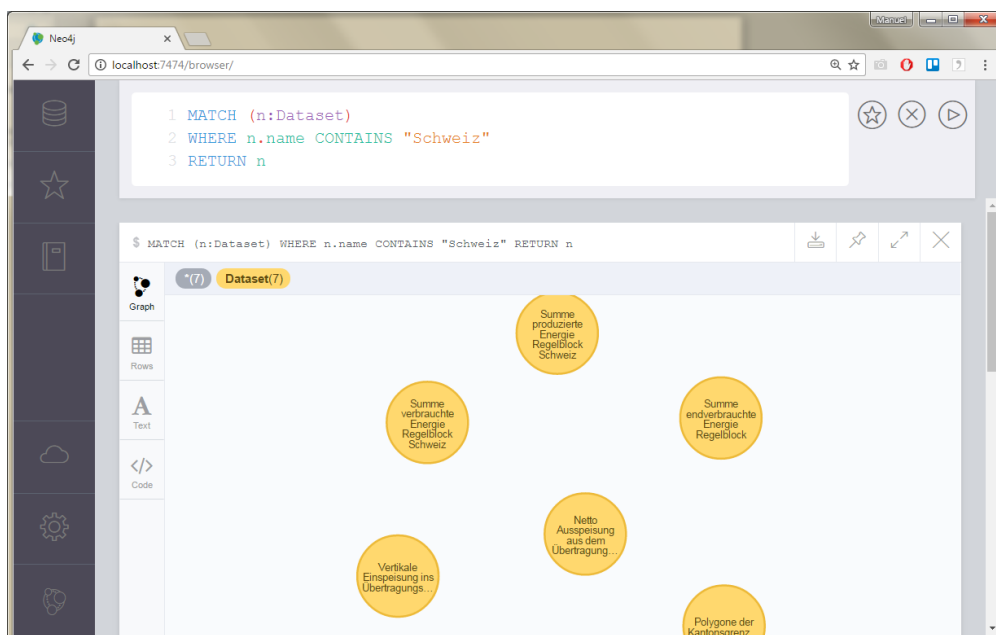


Abbildung B.1: Neo4j Browser mit Abfragebeispiel (Ergebnisse als Graph)

Über Cypher Befehle können neue Objekte in die Datenbank gespeichert, bestehende geändert oder alte Objekte gelöscht werden. Den vollständigen Funktionsumfang kann im Neo4j Benutzerhandbuch (Neo Technology, Inc., 2016f) nachgelesen werden.

Durch die Benutzung von Cypher ist diese Art der Datenbankverwaltung umständlich, da eine eigene Abfragesprache erlernt werden muss. Aus diesem Grund wurde für diese Bachelorarbeit zusätzlich ein „Neo4jHelper“ Script entwickelt.

Dieses benutzt das R-Paket RNeo4j, passt aber dessen Funktionen an das entwickelte Datenmodell an und stellt sie in der R Umgebung zur Verfügung.

B.1.1 Zeitreihendaten in Datenbank einfügen

Sollen neue Daten in die Datenbank einfügen werden, macht man dies am besten in einer R Umgebung. Darin lädt man das „Neo4jHelper“ Script, um dessen Funktionen zur Verfügung zu haben. Ausserdem verbindet man sich mit der Datenbank über die Funktion „startGraph“ von RNeo4j.

Die zu importierenden Daten müssen in R geladen und in die Form eines „data.table“ (Datenstruktur in R welches einer Tabelle gleicht) gebracht werden. Um ein TemporalData Objekt zu erstellen müssen folgende Daten enthalten sein:

- `datetime`: Messzeitpunkt im UTC Format, in Epoch Darstellung¹, numerischer Wert
- `value`: Messwert, numerischer Wert
- `unit`: Masseinheit, Text

Jetzt erstellt man einen neuen Datensatz über die Funktion „importDataset“. Diese benötigt die Argumente Name und Beschreibung des neuen Datensatzes. Als Rückgabewert erhält man den erstellten Node, welchen man in einer Variable ablegt.

Anschliessend kann durch die Daten iteriert werden. Pro Zeile, also pro Datenpunkt, ruft man die Funktion „importTemporalDataToDs“ auf. Als Argumente gibt werden ihr die Datenbankverbindung, den Datensatz-Node, `datetime`, `value` und `unit` mitgegeben.

Danach können die Daten über die Funktion „importTemporalDataToDs“ importiert werden. Die Funktion benötigt die Argumente Datenbankverbindung, Datensatznode, `datetime`, `unit` und `value`.

Sollten die Daten lokalisiert sein, lässt sich das mit „connectDatasetToSpatial“ in die Datenbank eintragen. Die Funktion muss mit Datenbankverbindung, Name des zu lokalisierende Datensatzes, Name des Datensatzes des SpatialObjects und Name des SpatialObjects aufgerufen werden.

Mit „assignTimeTree“ lässt sich nun der Messabstände der Daten festlegen (siehe Abbildung B.2). Dies geschieht, indem eine Relation zum jeweiligen „TimeTree“-Objekt erstellt wird. Die Funktion nimmt als Argumente die Datenbankverbindung, den Namen des Datensets und den Messabstand als Text (erlaubte Werte sind in Abbildung B.2 ersichtlich).

¹ Epoch ist eine numerische Darstellungsform eines Zeitpunktes, wird in Sekunden ab dem 1. Januar 1970, 00:00 Uhr UTC angegeben.

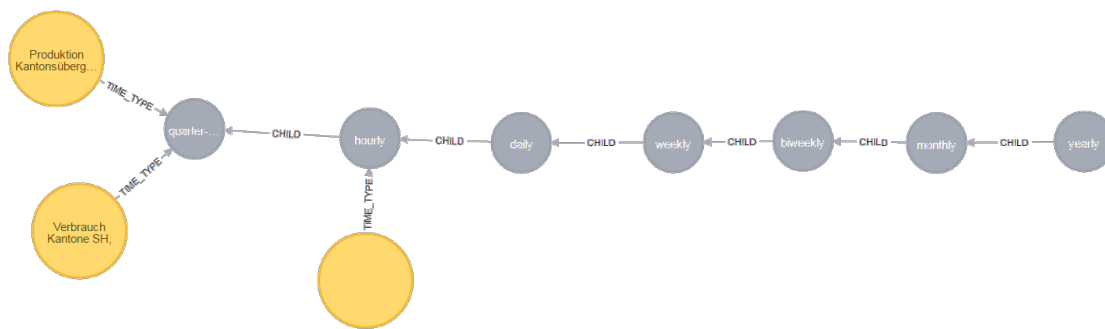


Abbildung B.2: TimeTree mit Beispieldatensätzen

Über die Funktion „assingTimeObjects“ können die Datensets mit TimeObjects verbunden werden, welche die enthaltene Zeitspanne beschreiben. TimeObjects existieren in drei Typen (Jahr, Woche, Wochentag). Ein Datensatz mit Informationen zu Kalenderwoche 1 im Jahr 2016 sollte also mit den TimeObjects [Jahr:2016], [Kalenderwoche:1] verbunden sein. Um diese Verbindungen zu erfassen, müssen vorgängig drei Listen erstellt werden. Es ist jeweils eine für Jahre, Kalenderwochen und Wochentage notwendig. Wochentage müssen dabei als englische Texte vorliegen, Wochen und Tage als numerische Werte. Nun kann die Funktion mit Argumenten Datenbankverbindung, Datensatz-Node und den drei Listen aufgerufen werden.

```
# load RNeo4j
library(RNeo4j)
# import all helper functions
source("Neo4jHelper.R")

# Connect to DB
graph <- startGraph("http://localhost:7474/db/data/", "neo4j", "Neo4j")

data <- data.frame(datetime = c(1474638521, 1474637458),
                    value = c(3.2334, 5.66743),
                    unit = c("MWh", "MWh"))

# data:
#   datetime  value  unit
# 1474638521 3.2334  MWh
# 1474637458 5.66743 MWh

# import new dataset
ds.node <- importDataset(graph, "neues Datenset", "neue Daten")

# iterate over data and import it
for (i in seq_len(nrow(data))) {
  importTemporalDataToDs(graph, ds.node,
                        datetime = data[i,1],
                        value = data[i,2],
                        unit = data[i,3])
}
```

Programcode B.1: Beispiel eines Zeitreihenimports

B.1.1.1 Zusatzinformationen

- Will man Eigenschaften importieren, können der Funktion „importTemporalDataToDs“ zwei zusätzliche Argumente mitgegeben werden: „is.property“ = TRUE und der Name der Eigenschaft.
- Will man zusätzliche Daten in ein bereits vorhandenes Datenset importieren, kann man den dafür notwendigen Node über die Funktion „getDatasetNode“ finden.
- Importiert man grosse Datenmengen, wie zum Beispiel die Swissgrid Netzdaten eines ganzen Jahres, lohnt sich ein CSV Import über eine Cypher Query. Zuerst wird jeder Datensatz als CSV im „Import“-Verzeichnis von Neo4j gespeichert. In Cypher kann man mit „LOAD CSV“ auf diese Dateien zugreifen. Ein Beispiel findet sich in ReadExcelIntoNeo4j.R.

B.1.2 Geodaten / Shapefiles in Datenbank einfügen

Geodaten können als Shapefile in die Datenbank importiert werden. Hierfür findet sich im Neo4jHelper Script die Funktion „importShpfile“. Diese Funktion erstellt ein neues Datenset und importiert alle im Shapefile enthaltenen Geoobjekte. Folgende Argumente werden benötigt:

- graph: Datenbankverbindungsobjekt, mit Funktion „startGraph“ von RNeo4j erzeugt
- filename: Name des Shapefiles (ohne Endung)
- filepath: absoluter Dateipfad zum Shapefile
- ds.name: Name des zu erstellenden Datensets
- ds.description: Beschreibung des zu erstellenden Datensets
- datasource: Node einer Datenquelle

Da SpatialObjects einen Namen benötigen, sucht die Funktion in den Metadaten nach einem Attribut, welches die Buchstabenfolge „name“ enthält. Findet sich eines, wird dieses als Name des SpatialObjects benutzt. Die restlichen Attribute werden nicht importiert.


```

# Import Shapefiles -----
# create datasource
ds.source <- createDatasource(graph, "Swiss Topo", "Swiss Topo, BAFU")

# shapefiles located in ../workingdirectory/shapefile/
filepath <- paste0(getwd(), "/shapefile")

shp.filesnames <- c("ARA_2014_SWW",
                   "KVA4_geocoded_ch")

shp.dsname <- c("Abwasserreinigungsanlagen",
               "Kehrichtverbrennungsanlagen")

shp.dsdesc <- c("Positionen der schweizer Abwasserreinigungsanlagen",
               "Positionen der schweizer Kehrichtverbrennungsanlagen")

for (i in 1:length(shp.filesnames)) {
  importShpfile(graph, shp.filesnames[i], filepath, shp.dsname[i], shp.dsdesc[i], ds.source)
}

```

Programcode B.2: Beispiel eines Imports von 2 Shapefiles

B.1.3 Daten löschen

Zum Löschen von Daten wird der Neo4j Browser benutzt, da dort mittels Cypher Queries schnell und einfach Objekte gelöscht werden können. Um sicherzustellen, dass die richtigen Objekte gelöscht werden, sollten diese in einem ersten Schritt im Browser kontrolliert werden. Dies macht man, indem man sich die Objekte mit einer Query ausgeben lässt („RETURN“ in letzter Zeile). Jetzt kann man die Query verändern, bis ausschliesslich die zu löschenden Objekte angezeigt werden.

Hat man die Objekte kontrolliert, ersetzt man „RETURN“ durch „DETACH DELETE“. Dies löscht alle Objekte inklusive deren Verbindungen innerhalb der Datenbank.

```

MATCH (n:Dataset)-[:CONTAINS]-(m)
WHERE n.name = "zu löschendes Datenset"
RETURN n, m

```

Programcode B.3: Cypher Abfrage um Objekte im Neo4j Browser zu kontrollieren

```

MATCH (n:Dataset)-[:CONTAINS]-(m)
WHERE n.name = "zu löschendes Datenset"
DETACH DELETE n, m

```

Programcode B.4: Cypher Abfrage um Objekte im Neo4j Browser zu kontrollieren

B.2 Visualisierungstool

Soll mit dem Visualisierungstool eine Grafik erzeugt werden, besteht dieser Arbeitsschritt aus 3 Schritten verteilt auf zwei Tabs:

- Datenauswahl (Tab Database Overview)
- Konfigurieren der Visualisierung (Tab Visualization)
- Export (Tab Visualization)

Das Tool unterstützt zwei Zusatzfunktionen: Erstens kann im Tab Database Exploration nach verwandten Datensets gesucht werden. So können Informationen gefunden werden, welche für die visualisierten Datensätze eventuell von Relevanz wären. Zweitens können die geladenen Daten im Tab Raw Data im Rohformat betrachtet werden, um so Ausreisser oder spezielle Vorkommnisse näher zu analysieren.

In Anhang A sind vollständige Screenshots jedes Tabs angefügt (Abbildungen A.1, A.2, A.3, A.4 sowie A.5). Aufgrund der Grösse können diese nicht in den Text eingebunden werden.

B.2.1 Tab Data Overview

In diesem Tab können Datensätze aus der Datenbank geladen werden. Um diese zu finden, können ein Textfilter sowie ein zeitlicher Filter benutzt werden, um die Suchresultate einzuschränken.

Der Textfilter unterstützt mehrere Suchtexte, indem die einzelnen Suchbegriffe mit Semikolon abgetrennt werden (siehe Abbildung B.3). Der Filter wird automatisch angewendet. Wird das Textfeld leergelassen, deaktiviert sich der Filter.

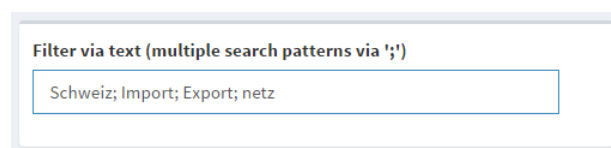


Abbildung B.3: Beispiel eines angewendeten Textfilters

Als zweiter Filter steht ein zeitlicher Filter bereit, der in einem aufklappbaren Panel mit Titel „Temporal Filter“ zu finden ist (siehe Abbildung B.4). Dieser hat drei Funktionen: Erstens kann er nach Start- und Enddatum, zweitens nach Kalenderwochen und drittens nach Wochentagen filtern. Start- und Enddatum werden über ein Kalender-Popup ausgewählt, Kalenderwochen werden als kommagetrennte Liste erfasst und Wochentage werden mittels Checkboxes ausgewählt.

Mit Klick auf „Apply“ werden die Filter angewendet. Dabei kann der Kalenderwochenfilter und der Wochentagsfilter deaktiviert werden, indem der Text leergelassen beziehungsweise alles abgewählt wird. Der Start- und Endfilter wird in jedem Fall aktiviert. Im Anschluss werden alle Datensätze angezeigt, welche die Filterbedingungen erfüllen. Die Filterung kann über den „Reset“-Button gelöscht werden.

Temporal Filter

date interval filter: start and enddate
12.07.2011 to 24.09.2016

calendar week filter (e.g. 15, 16)
1, 2, 3, 4, 5, 6, 7

weekday filter:
☒ Monday
☐ Tuesday
☐ Wednesday
☐ Thursday
☒ Friday
☒ Saturday
☒ Sunday

apply filter delete filters

Abbildung B.4: ausgefüllter zeitlicher Filter im ausklappbaren Panel

Datensätze werden geladen, wenn darauf geklickt wird. Nach einigen Sekunden aktualisiert sich die Tabelle unterhalb der Graph-Visualisierung und zeigt Informationen zum eben ausgewählten Datenset an (siehe Abbildung B.5). Sollten ein oder mehrere Datensets doch nicht benötigt werden, können diese mit einem Links-Klick markiert und über den „Delete selected Datasets“-Button gelöscht werden.

Show entries

Search:

	Name Datenset	Beschreibung Datenset	Anzahl Zeitserien-Daten	Anzahl Geo-Objekte	Anzahl Event-Daten	Lokalisierung
	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>	<input type="text" value="All"/>
1	Summe endverbrauchte Energie Regelblock Schweiz	Summe endverbrauchte Energie Regelblock Schweiz	35040	0	0	true
2	Summe produzierte Energie Regelblock Schweiz	Summe produzierte Energie Regelblock Schweiz	35040	0	0	true
3	Kantons Grenzen Schweiz	Polygone der Kantons Grenzen in der Schweiz	0	26	0	false

Showing 1 to 3 of 3 entries

Previous Next

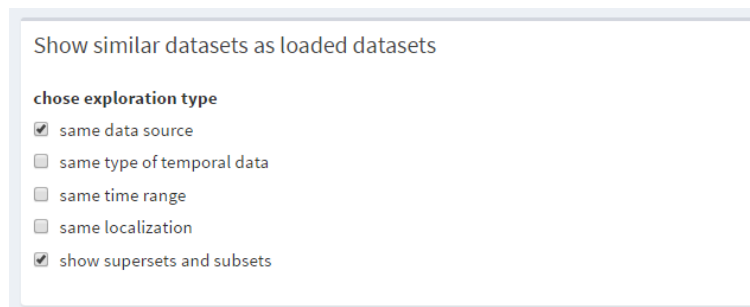
Abbildung B.5: Übersicht geladene Daten mit zwei zum Löschen markierten Datensätzen

B.2.2 Tab Data Explorer

Im Data Explorer kann man ähnliche Datensets finden. So können noch nicht berücksichtigte Daten gefunden werden, welche in die Visualisierung eingebaut werden sollten oder Hintergrundinformationen enthalten.

Im Tool werden fünf Arten zur Suche von ähnlichen Datensets angeboten (siehe Abbildung B.6):

- Gleiche Datenquelle: Zeigt alle Datensätze mit derselben Datenquelle
- Gleiche Zeitreihenart: Zeigt alle Datensätze, welche denselben Messabstand zwischen Zeitwerten besitzen (bspw. 15-Minütlich)
- Gleiche Zeitspanne: Zeigt alle Datensätze, welche Daten in demselben Zeitraum enthalten (z.B. gleiche Kalenderwoche, gleiches Jahr, etc.)
- Gleiche Lokalisierung: Zeigt alle Datensätze, welche dieselbe Lokalisierung aufweisen
- Super- und Subsets anzeigen: Zeigt alle Supersets, alle Datensätze, welche in diesen Supersets enthalten sind und alle Subsets der geladenen Daten an



Show similar datasets as loaded datasets

chose exploration type

- ☒ same data source
- ☐ same type of temporal data
- ☐ same time range
- ☐ same localization
- ☒ show supersets and subsets

Abbildung B.6 Auswahl der Methoden um ähnliche Datensets zu finden

Sind eine oder mehrere Methoden gewählt, aktualisiert sich der darunterliegende Graph (siehe Abbildung B.7). Dieser hat eine Baumstruktur, wobei die bereits geladenen Datensets auf der linken Seite ersichtlich sind, während die nun gefundenen auf der rechten Seite zu finden sind. Dazwischen werden jene Nodes angezeigt, welche die Daten verbinden. Anhand der Pfeilbeschriftungen kann man erkennen, durch welche Methode die Datensets gefunden wurden.

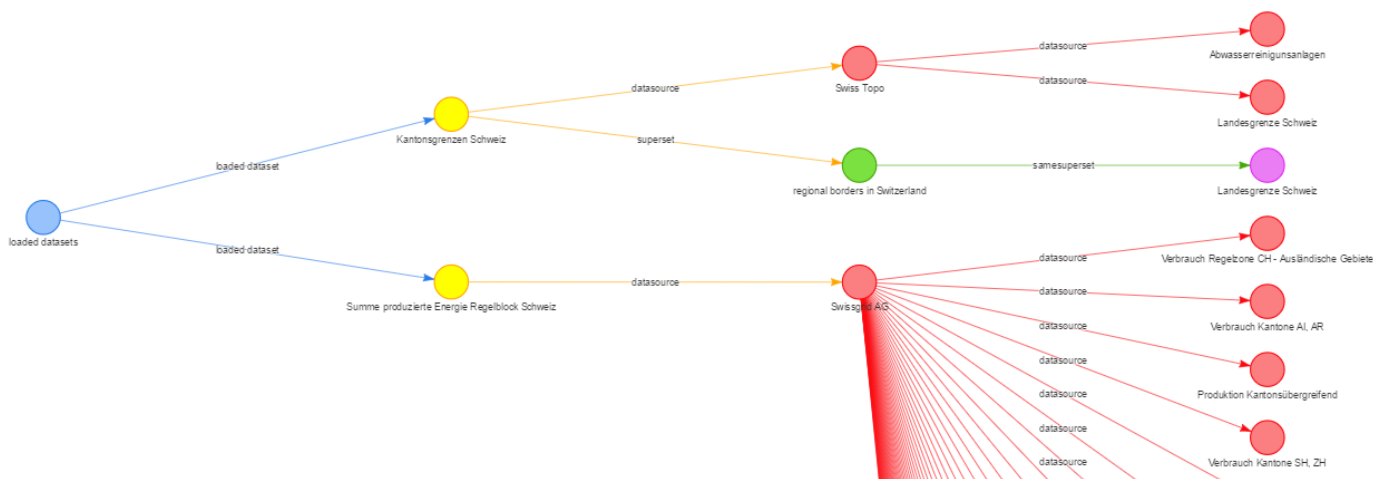


Abbildung B.7 Data Exploring mit zwei aktivierten Suchmethoden

B.2.3 Tab Visualization

In diesem Tab werden standardmässig zwei Visualisierungen angezeigt. Im oberen Bereich befindet sich ein Zeitreihenchart, welcher bereits standardmässig alle geladenen Zeitreihen enthält. Sind lokalisierte Daten oder Geoobjekte geladen, werden diese unterhalb der Zeitreihen auf einer Karte dargestellt. Die Farbe, mit welcher sie angezeigt werden, wird pro Datensatz bestimmt. So ist stets ersichtlich, welche Zeitreihenelemente zu welchen Geoobjekten gehören.

Die interaktive Darstellung der Zeitreihen ermöglicht es, den angezeigten Ausschnitt anzupassen. Ausserdem erscheint beim Hovern über Daten ein Informations-Popup, welches Details zu den Messpunkten anzeigt. Die Visualisierung kann zu jedem Zeitpunkt exportiert werden. Dazu kann in den Bedienelementen der Visualisierung „download plot as png“ geklickt werden. Es öffnet sich ein Download-Dialog, in welchem man Speicherort der Datei angeben kann.

Die Kartendarstellung ist auch interaktiv und kann durch Zoomen erforscht werden. Der Export steht hier als Button „download map“ zur Verfügung. Hier erhält man eine PDF-Datei, welche die herausgezoomte Ansicht der Karte enthält.

Nun kann die Visualisierungen konfiguriert werden. Dazu hat man folgende Möglichkeiten:

- Datensets und deren Bestandteile ein- / ausblenden
- Berechnete Datensets erstellen
- Code des Zeitreihencharts anpassen
- Code der Karte anpassen

Im oberen Bereich des Tabs befindet sich die „Show / Hide“-Box, um Datensets ganz oder teilweise auszublenden (siehe Abbildung B.8). Dazu wird für jedes Datenset ein aufklappbares Panel generiert, in welchem per Checkbox Teile ausgeblendet werden können.

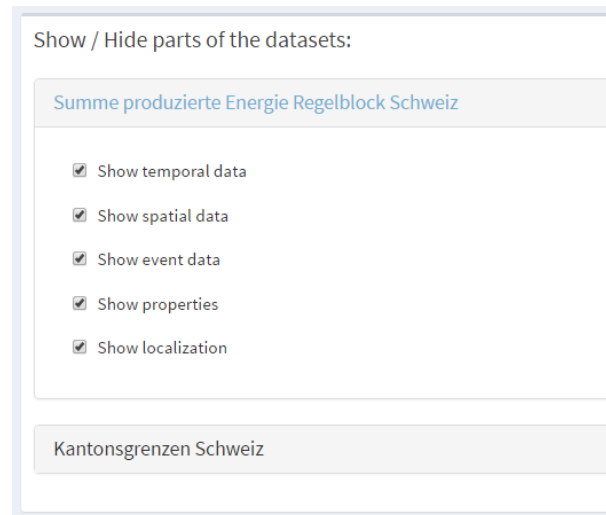


Abbildung B.8: aufklappbare Panels, in welchem Datensetteile ausgeblendet werden können

Unterhalb der in Abbildung B.8 gezeigten Box können berechnete Datensets erzeugt werden. Durch Klick auf den Button „Modify code to create calculated datasets“ öffnet sich ein modaler Dialog mit einem Code Editor (siehe Abbildung B.9). Dieser Code wird bei Klick auf den „Apply“-Button direkt ausgeführt. Neben den Funktionen aller geladenen Libraries stehen auch die Funktionen des Neo4jHelper Scripts zur Verfügung.



Abbildung B.9: Code Editor, mit welchem berechnete Datensets erzeugt werden können

Auf die gleiche Art kann die Zeitreihenvisualisierung geändert werden. Mit Klick auf den Button „modify plot code“, welcher sich unterhalb der Visualisierung befindet, öffnet sich der gleiche Code Editor. Dieser enthält zu Beginn den Code der Standardeinstellungen (siehe Programmcode B.6). Dort wird mittels ggplot2 ein „gg“ Objekt erstellt, welches die Zeitreihendarstellung beinhaltet. Durch Anpassen dieses Objekts kann die Visualisierung verändert werden.

```
gg <- ggplot()
# if temporaldata is available
if (length(data$td) > 0) {
  # go through all datasets
  td.ds <- distinct(data$td)[[1]]
  for (i in 1:length(td.ds)) {
    # get temporal data of dataset
    line.data <- filter(data$td, IDds == td.ds[i])
    # add td as line to gg
    gg <- gg + geom_line(aes(td.datetime, td.value),
                        alpha = 0.5, colour = settings$pal(td.ds[i]),
                        data = line.data)
  }
}
# convert ggplot2 object into interactive plotly object
ggplotly(gg)
```

Programmcode B.5: Standardcode für Zeitreihenvisualisierung

Auch die Kartendarstellung kann auf gleiche Weise verändert werden. Hier werden im Standardcode (siehe Programmcode B.7) alle Polygone von Geodaten und Lokalisierungen hinzugefügt.

```

# Create leaflet map with CartoDB.Positron background images
map <- leaflet() %>% addProviderTiles('CartoDB.Positron')
# add polygon geoobjects of loaded datasets
if (!is.null(mapObjects$polygons)) {
  map <- map %>%
    addPolygons(data = mapObjects$polygons, color = ~settings$pal(IDds),
  fillColor = ~settings$pal(IDds))
}
# add localization polygons
if (!is.null(mapObjects$loc_polygons)) {
  map <- map %>%
    addPolygons(data = mapObjects$loc_polygons, color = ~set-
  tings$pal(IDds), fillColor = ~settings$pal(IDds))
}
# return map
map

```

Programmcode B.6: Standardcode für Kartendarstellung

B.2.4 Tab Raw Data

Im Tab Raw Data kann der Benutzer die Rohdaten aus der Datenbank betrachten. Hier findet man fünf Tabellen, in der die aus der Datenbank geladenen Rohdaten mit allen Eigenschaften dargestellt werden. Die Daten werden anhand des im Visualisierungstools eingesetzten Datenmodells aufgeteilt (Details siehe Abschnitt 6.4.1). Anhand der IDs kann man erkennen, welche Daten aus welchem Datensatz stammen. Folgende Tabellen werden angezeigt:

- Datensätze
- Zeitreihendaten
- Geo-Objekte
- Event-Daten
- Lokalisierungsdaten

Anhang C

Liste der Verbesserungen

C.1: Funktionalitäten

- Datenbankverwaltung:
 - Kommentare: Kommentare in der Datenbank sollen im Visualisierungstool angezeigt werden, neue sollen erfasst werden können.
 - Berechnete Datensätze: Berechnete Datensätze sollen der Benutzer abgespeichern können.
 - Aufräum-Script: Es soll ein Script entwickelt werden, mit welchem überprüft wird, ob die Datensätze mit allen relevanten TimeObjects verbunden sind. So wird die Datenintegrität sichergestellt.
- Arbeitsstand speichern: Der Arbeitsstand soll zwischengespeichert werden können.
- Koppelung von Karte und Zeitreihe: Wird auf einer Visualisierung etwas ausgewählt, sollen Daten desselben Datensets auch auf der anderen Visualisierung hervorgehoben werden.
- Geo-Filterung: Der Benutzer soll Datensätze nach Geoinformationen filtern können.
- Geographische Analyse zeitlicher Daten: Zeitreihendaten sollen auf einer animierten Karte dargestellt werden können, sodass Zeitreihedaten als sich verändernde Punkte angezeigt werden.
- Mehrere Visualisierungen: Um schnellere Vergleiche zu ermöglichen, sollen mehrere Visualisierungstabs zur Verfügung stehen.
- Deployment: Das Visualisierungstool sollte auf einem Server installiert und über das Netzwerk zugänglich gemacht werden

C.2 Refactoring

Neben fehlenden Funktionen sollten folgende Bereiche optimiert werden:

- Neo4jHelper Script: Das Script sollte überarbeitet werden, da einige Funktionen nicht mehr benötigt werden.
- Bugfixing: Es sollte nach Bugs gesucht und diese ausgebessert werden.
- Layout: Die Darstellung einiger Elemente sollte verbessert werden

Inhalt der CD

Folgende Dateien befinden sich auf der CD:

- deutsche Zusammenfassung (Zusfsg.txt)
- Englische Zusammenfassung (Abstract.txt)
- Bachelorarbeit als PDF-Datei (Bachelorarbeit.pdf)
- Applikationscode und Scriptcode in einem ZIP-Archiv (Code.zip)
- Beispieldatenbank in einem ZIP-Archiv (Datenbank.zip)