Beuth Hochschule für Technik Berlin
Fachbereich VI – Informatik und Medien

Konzeption und Umsetzung einer Web-Applikation zur visuellen Exploration von Multikollektivität

BACHELORARBEIT

Zur Erlangung des akademischen Grades *Bachelor of Science (B.Sc.)*

vorgelegt von: Nadja Zollo
Matrikelnummer 763962

1. Betreuer: Prof. Dr.-Ing. Hartmut Schirmacher

2. Betreuerin: Prof. Dr. Stefanie Rathje

Gutachterin: Prof. Dr. oec. Petra Sauer

Abgabedatum: 10.09.2013

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kapitel 1

Einleitung

Es gibt ein frühes, in seiner Schlichtheit sehr beeindruckendes Beispiel von Informationsvisualisierung aus dem Jahre 1854. Dabei handelt es sich um eine Karte des Londoner Stadtbezirkes Soho, in dem zu diesem Zeitpunkt eine Choleraepidemie ausgebrochen war. Auf dieser Karte sind die cholerabedingten Todesfälle sowie die Standorte von Wasserpumpen markiert. Der damalige Amtsarzt von London, Dr. John Snow, konnte anhand der Anordnung der Markierungen eine Anhäufung von Todesfällen im Umkreis einer Wasserpumpe in der damaligen Broad Street erkennen. Seine darauffolgende Verfügung, diese Wasserpumpe für die Öffentlichkeit zu sperren, dämmte neue Choleraerkrankungen ein und bewirkte eine Abnahme der durch Cholera verursachten Todesfälle [vgl. Spe07, S. 3] (siehe auch 2.2).

Wie in diesem Beispiel für die Medizin ist das Erkennen von Strukturen, Beziehungen, Mustern und Zusammenhängen noch für viele andere Disziplinen – wie etwa Biologie, Geographie, Kriminologie, Sozial- und Kulturwissenschaften, Klimaforschung, Wirtschafts- und Finanzwesen – wichtig und von großer Bedeutung für das Verständnis verschiedenster Sachverhalte. Hierbei leistet nicht zuletzt die adäquate visuelle Darstellung der Daten bzw. der in den Daten enthaltenen Informationen einen wertvollen Beitrag zu einem Erkenntnisgewinn. Die Visualisierung von Informationen kann Aufschluss über etwas geben, was allein mit der Abbildung bloßer Daten, beispielsweise Zahlen in einer Tabelle, zunächst nicht vermutet oder offensichtlich wird. In diesem Zusammenhang kommt einem unweigerlich die bekannte Redensart „Ein Bild sagt mehr als tausend Worte.“ in den Sinn, um die zuvor gemachte Aussage zu bestätigen und zu untermauern [vgl. Fek+08, S. 4–6].

Allerdings begünstigt nicht jede Visualisierung automatisch den Prozess der Informationsverarbeitung im menschlichen Gehirn. Neben den Fähigkeiten zur Wahrnehmung und Kognition, die bei jedem Menschen unterschiedlich ausgeprägt sein können, sind auch die Art und die Gestaltung der Visualisierung maßgeblich für eine erfolgreiche Darstellung und Interpretation der Daten und Informationen verantwortlich. Je nach Datensatz und Problemstellung ist daher

eine sorgfältige Auswahl der Visualisierungstechnik geboten. So unterscheidet sich ein Diagramm zur Darstellung von Hierarchien (z.B. Führungsstrukturen in einem Unternehmen) von einem Diagramm, das die Entwicklung eines Aktienkurses an der Börse veranschaulicht [vgl. BOH10].

Ein wichtiges Element bei der computergestützten Visualisierung von Informationen ist die interaktive Exploration. Stellt man die Funktionalität bereit, bestimmte Teilaspekte oder Details der Datenmenge gesondert darzustellen, beispielsweise mittels Setzen eines Filters, kann dadurch ein genaueres Verständnis der Dateneigenschaften und Zusammenhänge erlangt werden. Idealerweise verstärkt dies den allein durch die graphische Abbildung von Daten bereits entstandenen „Aha!-Effekt“ [vgl. Spe07, S. 5] und kann gleichzeitig auch zu einer besseren Akzeptanz der Ergebnisse beitragen sowie zu einer weitergehenden Erkundung und neuen Fragestellungen motivieren.

Dank der bisherigen Errungenschaften auf dem Forschungsgebiet der Informationsvisualisierung wie auch durch die technologischen Fortschritte und Entwicklungen in der Informatik ist es heutzutage möglich, etablierte Techniken und Methoden mit modernen Technologien und Anwendungen zu kombinieren.

1.1 Motivation und Zielsetzung

Wie zuvor bereits erwähnt, wird die Visualisierung von Informationen auch in den Sozial- und Kulturwissenschaften eingesetzt und so ist das Thema der vorliegenden Bachelorarbeit im Kontext einer Projekt-Kollaboration zwischen Frau Prof. Dr. Stefanie Rathje und Herrn Prof. Dr.-Ing. Hartmut Schirmacher entstanden.[1] Letzterer ist Professor für Computergrafik und Programmierung an der Beuth Hochschule für Technik Berlin, Frau Prof. Dr. Stefanie Rathje ist Professorin für Unternehmensführung und Kommunikation an der Hochschule für Technik und Wirtschaft Berlin und forscht unter anderem zu Interkultureller Kommunikation, Interkulturellen Kompetenzen sowie zu zeitgemäßen Kultur- und Kollektivkonzeptionen.

Multikollektivität (siehe Abschnitt 2.1) ist solch ein modernes Konzept in den Kulturwissenschaften, nach dem jeder Mensch vielen verschiedenen Kollektiven gleichzeitig zugehören kann. Die kulturellen Einflüsse dieser Gruppen haben in Verbindung mit seiner Herkunft eine Wirkung auf das Individuum und formen dessen Identität. Die Gruppenzugehörigkeiten können dabei teilweise temporär und sehr dynamisch sein. Gleichwohl bleiben viele Menschen in der Annahme verhaftet, einer einzigen Kultur anzugehören und nur durch diese geprägt

[1] Die Projektskizze befindet sich im Anhang auf Seite xii.

zu sein. Dies wirkt sich in interkulturellen Trainings dadurch aus, dass die Teilnehmenden zunächst ausschließlich das vermeintlich trennende – die „andere“ Kultur – wahrnehmen. Das Sichtbarmachen von Gemeinsamkeiten, nämlich die Zugehörigkeit zu derselben Gruppe unabhängig von der nationalen oder ethnischen Abstammung (z.B. Studierende, Liebhaber einer bestimmten Musik- oder Kunstrichtung, Hobbykoch, Fußballfan etc.) bietet hier die Chance, den Blickwinkel zu verändern, um die Eigenschaften zu bemerken, die die Individuen miteinander verbinden.

Vor diesem kulturwissenschaftlichen Hintergrund und als erster Beitrag zu dem genannten Projekt ist das Ziel der vorliegenden Bachelorarbeit die Konzeption und die Umsetzung einer Web-Applikation, die es ermöglicht, Gemeinsamkeiten und Relationen innerhalb einer Gruppe zu untersuchen und zu visualisieren. Grundlage für die Analyse und die Visualisierung bilden Begriffe, welche die Individuen der Gruppe durch Interaktion mit der Anwendung eingeben und die in einer Datenbank erfasst werden.

Unter Berücksichtigung der Konzepte und Methoden der Informationsvisualisierung sowie moderner Web-Technologien soll ein Visualisierungs-Werkzeug entstehen, das für die Benutzerinnen und Benutzer einen optimalen Erkenntnisgewinn erzeugt, indem entsprechende graphische Darstellungen Aufschluss über Gemeinsamkeiten und Zugehörigkeiten geben. Hierbei kommen verschiedene Betrachtungsmodi zum Tragen, sodass es beispielsweise möglich sein soll, sich die Relationen zwischen zwei Individuen sowie auch innerhalb der gesamten Gruppe anzeigen zu lassen.

Der Schwerpunkt der Arbeit liegt auf der Erforschung und dem Testen möglicher Verfahren und Technologien zur Entwicklung eines ersten Prototypen der gewünschten Web-Applikation. Hierzu zählt auch, eine tragfähige und erweiterbare Applikationsarchitektur zu konzipieren. In diesem Sinne wird es in den nachfolgenden Kapiteln um die Beantwortung der Frage gehen, welche Anforderungen eine Web-Applikation erfüllen muss, damit sie zur visuellen Exploration von Multikollektivität eingesetzt werden kann sowie um die Aufgabe, die richtigen Mittel und Methoden zur Umsetzung einer solchen Anwendung zu finden.

1.2 Abgrenzung

Wie bereits beschrieben ist eine Untersuchung der von den Benutzerinnen und Benutzern eingegebenen Begriffe hinsichtlich ihrer Übereinstimmungen erforderlich. Da jedoch semantisches Begriffsmatching und die Realisierung eines automatisierten Verfahrens zur Gewichtung der Begriffsnähe bzw. -verwandschaft im Rahmen dieser Arbeit nicht im Fokus stehen, wird die

Suche nach Gemeinsamkeiten auf Begriffe beschränkt, die gleich sind. Nichtsdestotrotz stellen semantische Technologien für die Weiterentwicklung des Prototyps und die Fortführung des Gesamtprojekts sowie für Anschlussarbeiten interessante Möglichkeiten dar.

1.3 Aufbau der Arbeit

In Kapitel 2 *Fachliches Umfeld* werden die theoretischen Grundlagen und Methoden der Fachgebiete vorgestellt, die in diese Arbeit einfließen. Zunächst befasst sich Abschnitt 2.1 mit den kulturwissenschaftlichen Aspekten rund um den Begriff *Multikollektivität*, gefolgt von Ausführungen zur *Informationsvisualisierung* in Abschnitt 2.2. Neben allgemeinen Erläuterungen kommt dem Abschnitt 2.2.2 *Visualisierung von Relationen* eine wichtige Rolle zu, in dem es um die für diese Arbeit zentralen Eigenschaften von Graphen und Netzwerken geht und eine Übersicht relevanter Visualisierungstechniken geboten wird. Abschnitt 2.3 beschreibt die *Technologien*, die zur Realisierung einer modernen Web-Applikation herangezogen werden sollen und beschäftigt sich zudem mit der Wahl einer geeigneten *Datenbank* in Abschnitt 2.3.3.

Das Kapitel 3 *Konzeption und Entwurf der Applikation* behandelt einen wesentlichen und entscheidenden Teil der vorliegenden Bachelorarbeit. Hier werden unter anderem die grundlegenden Anforderungen und das Datenmodell ausgearbeitet sowie Überlegungen zur Benutzeroberfläche und zu den Abläufen erörtert – dies dient als Grundlage für die *Technische Umsetzung* der entwickelten Ideen in Kapitel 4.

Anschließend wird im 5. Kapitel *Ergebnisse und Auswertung* dargestellt, welche Lösungsideen wie gut funktionieren und wo sich Stärken und Schwächen der Anwendung erkennen lassen, bevor durch das Kapitel 6 *Fazit und Ausblick* eine Bewertung der Ergebnisse erfolgt und Überlegungen zu potentiellen Erweiterungen angestellt werden.

Kapitel 2

Fachliches Umfeld

Diesem Kapitel kommt die Aufgabe zu, die dem Thema dieser Bachelorarbeit zugrunde liegenden theoretischen Konzepte sowie wesentliche Methodiken und Verfahren zu erläutern. Ebenfalls wird es darum gehen, die technischen Aspekte zu betrachten und die erforderlichen Werkzeuge vorzustellen.

2.1 Multikollektivität

In diesem Abschnitt geht es um den kulturwissenschaftlichen Gegenstand, der den Hintergrund dieser Bachelorarbeit bildet. Vornehmlich soll es um die Vorstellung von *Multikollektivität* als Konzept in den modernen Kulturwissenschaften gehen und um die Bedeutung für den Bereich „Interkulturelle Trainings“.

2.1.1 Das Konzept der Multikollektivität

Der Begriff Multikollektivität wurde vor über 10 Jahren von dem Kulturwissenschaftler Klaus-Peter Hansen hervorgebracht . Darunter ist ein kulturwissenschaftliches Konzept zu verstehen, das sich der Beziehung zwischen einem Individuum und seiner sozialen Umwelt widmet [vgl. Rat12]. Zugrunde liegt dem Konzept eine Abkehr von der traditionellen Auffassung einer Teil-Ganzes-Beziehung, nach der ein Individuum fest in seiner sozialen Umwelt den dortigen kulturellen Einflüssen ausgesetzt ist. Stattdessen lässt sich mit dem Konzept Multikollektivität die Beobachtung beschreiben, „dass jedes Individuum gleichzeitig Teil zahlreicher Kollektiv- und damit auch Kulturzusammenhänge ist“ [Rat13], dabei aber seine Individualität bewahrt bleibt bzw. sich diese insbesondere durch die vielfachen Zugehörigkeiten herausbildet. Eindrücklich wird das Konzept, wenn man menschliche Kollektive im Vergleich zu Tierkollektiven beleuchtet und feststellt, das Multikollektivität eine wesentlich menschliche Eigenschaft ist, da

Tiere keine Möglichkeit haben, sich ihr Kollektiv (oder gar mehrere) auszusuchen, stattdessen daran gebunden sind [vgl. Rat13].

Eine weitere Definition des Konzepts Multikollektivität und eine Beschreibung charakteristischer Merkmale lässt sich anhand der nachfolgenden Aspekte durchführen [vgl. Rat12; Rat13]:

Mehrfachzugehörigkeit Die vielfachen Kollektivzugehörigkeiten eines Individuums sind ein elementares menschliches Merkmal. Hervorzuheben ist dabei, dass diese Vielfalt die Individualität des Einzelnen ausmacht. Dies ist eine moderne Sichtweise auf den Zusammenhang zwischen Kollektiven und Individuen, da nicht mehr davon ausgegangen wird, dass mit der Kollektivzugehörigkeit eine Anpassung einhergeht.

Individueller Überschuss Das Individuum gibt sich mit einer Zugehörigkeit zu einem Kollektivzusammenhang nicht auf oder diesem vollständig hin, sondern immer nur einen Teil seiner Person. Die unterschiedlichen Gewohnheiten in den Kollektiven, auf die das Individuum Zugriff hat, ermöglichen ihm eine differenzierte Sichtweise und stärken seine Autonomie.

Dialektik zwischen Kollektiven und Individuum Das Konzept der Multikollektivität ist kein simples Konzept, denn es vereint eine individuelle und eine kollektive Betrachtungsweise: zum einen geht es um das Individuum mit seinen Zugehörigkeiten und zum anderen um die mannigfach vorhandenen Kollektive, denen ein Individuum angehören kann.

Virulenz Das Konzept der Virulenz beantwortet die Frage, wie das Individuum mit den potentiell sehr unterschiedlichen Betrachtungsweisen und Gewohnheiten aus all seinen Kollektivzusammenhängen umgeht. Dabei geht es darum, dass das Individuum in der Lage ist, seine Zugehörigkeit zu einem Kollektiv bedarfsabhängig und je nach Situation anzuwenden und seine Identität zu bestimmen, ohne aufgrund von Widersprüchen in Konflikt zu geraten.

Kohäsion Menschliche Kollektive ziehen immer eine Grenze und sind ausschliessend. Allerdings sind es die mehrfachen Kollektivzugehörigkeiten der Individuen, die wiederum eine aufhebende Wirkung haben und ein stabiles Netzwerk bilden.

Das Konzept der Multikollektivität bietet in den Kulturwissenschaften neue Ansätze bei der Auseinandersetzung mit aktuellen Forschungsfragen und -themen. Insbesondere im Bereich der Interkulturalitätsforschung zum Thema *Interkulturelle Kompetenz* ergeben sich mit dem Konzept der Multikollektivität Chancen, ein moderneres Verständnis für den Begriff der Kultur

sowie den Einfluss von Kulturen auf die Individuen zu entwickeln und dieses u.a. im Rahmen von interkulturellen Trainings anzuwenden.

2.1.2 Neue Möglichkeiten für interkulturelle Trainings

Interkulturalität wird seit den 1990er Jahren verstanden als Zusammenstoß zweier Welten, zweier Kulturen. Gemeinhin geht man dabei davon aus, dass sich diese Kulturen voneinander abgrenzen lassen. Es wird ein Bild gezeichnet von einer einheitlichen Primärkultur (z.B. die Deutschen, die Franzosen), die das Wesen der Individuen bestimmt [vgl. Rat13]. Dieses Bild wird in interkulturellen Trainings zur Erklärung von individuellem Verhalten herangezogen, um die Teilnehmerinnen und Teilnehmer auf den Kulturschock vorzubereiten. Die Betonung der kulturellen Unterschiede führt jedoch dazu, dass sich Stereotypen intensivieren und sich Vorstellungen von klaren Kollektivgrenzen festigen. [vgl. Rat13].

Das Konzept der Multikollektivität vertritt an dieser Stelle den Ansatz, dass es aufgrund der Mehrfachzugehörigkeit immer zu einem Aufeinandertreffen mehrerer Welten kommt, wenn Individuen miteinander in Kontakt treten und demnach Interkulturalität etwas ist, was die Betroffenen konstruieren, da sie in der jeweiligen Situation keine gemeinsame Kollektivzugehörigkeit ausmachen können. Hieraus ergeben sich neue Aufgaben im Bereich der interkulturellen Trainings. Situationen, die bislang als interkulturell aufgeladen galten, müssen umgedeutet werden. Dies kann erreicht werden durch das Sichtbarmachen von gemeinsamen Zugehörigkeiten, die auf den ersten Blick aufgrund der vermeintlich vorherrschenden und trennenden Kollektivgrenzen nicht ins Bewusstsein rücken. Das Entdecken gemeinsamer Zugehörigkeit und damit verbunden die Vertrautheit gemeinsamer Gewohnheiten, schafft ein Gefühl von Zusammengehörigkeit und kann in Kollektivität überführt werden [vgl. Rat12; Rat13]. Für interkulturelle Trainings bzw. die interkulturellen Trainer ergibt sich somit die Möglichkeit wie auch die Anforderung, den Fokus auf die Vermittlung von „interkollektiver“ statt „interkultureller“ Kompetenz zu richten [vgl. Rat13].

2.2 Informationsvisualisierung

In der heutigen Zeit wird der Begriff „Visualisierung“ zumeist mit Computern oder Computersoftware und ähnlichen technologischen Aspekten in Verbindung gebracht. Das eingangs vorgestellte Beispiel aus dem Jahre 1854, zu sehen in Abbildung 2.1, demonstriert jedoch eine Visualisierung, die ganz ohne Computer erfolgt und verdeutlicht die eigentliche Definition des Wortes „visualisieren“, indem durch Sichtbarmachung eine Vorstellung, eine Idee

von etwas entwickelt werden kann und dies ein menschlicher Wahrnehmungsprozess ist [vgl. Spe07, S. 1-2]. Hierbei ist zu ergänzen, dass sich auch durch akustische Eindrücke oder andere Sinneserfahrungen Vorstellungswelten öffnen können und Visualisierung daher als eine Übersetzungsleistung zu verstehen ist, jegliche Art von Daten in einer geeigneten Form darzustellen, damit daraus eine Erkenntnis gewonnen werden kann [vgl. Spe07, S. 5; Sta13, S. 21]. Diese Leistung lässt sich noch genauer bestimmen, daher soll es im Folgenden darum gehen, das Konzept der Informationsvisualisierung zu erläutern und spezifische Methoden und Techniken herauszustellen.

Abbildung 2.1: Eine Variante der von John Snow angefertigten Grafik der Cholera-Epidemie in London/Soho 1854. Die Punkte lokalisieren Todesfälle, die Kreuze Standorte von Wasserpumpen (farbige Hervorhebung durch die Autorin, Quelle: [Sno])

2.2.1 Was ist Informationsvisualisierung?

Informationsvisualisierung hat die Aufgabe und das Ziel, durch die graphische Darstellung *abstrakter Daten* Menschen in ihrer Fähigkeit zur visuellen Wahrnehmung von Mustern, charakteristischen Eigenschaften und strukturellen Zusammenhängen zu unterstützen. Kennzeichnend für das Fachgebiet der Informationsvisualisierung (InfoVis) ist die computergestützte, interaktive, visuelle Repräsentation abstrakter Datenmengen im Vergleich zu anderen Teilgebieten wie *Scientific Visualization* (Erforschung und Visualisierung dreidimensionaler Erscheinungen) oder *Geovisualization* (Analyse und Visualisierung räumlicher Daten) [vgl. Shn03, S. 364-365; Spe07, S. 12-13].

Um das besonders ausgeprägte visuelle System des Menschen zur Exploration von Daten auszunutzen, kann man auf das sogenannte *Visual Information Seeking Mantra* zurückgreifen: „Overview first, zoom and filter, then details-on-demand“ [Shn03, S. 365]. Damit werden Richtlinien und Grundprinzipien bei der Informationsvisualisierung angesprochen. So soll zunächst ein Überblick (*Overview*) über die zugrunde liegende Datensammlung angeboten werden, gleichsam als Ausgangspunkt, um vom Allgemeinen mittels *Zoom*-Möglichkeit die Bereiche zu vergrößern, die von Interesse sind. Zusätzlich erlauben *Filter*, uninteressante Aspekte auszuklammern und den Fokus auf die gewünschten Informationen zu richten. Bei Bedarf werden dann Einzelheiten und weitere Informationen angezeigt (*details-on-demand*). Hinzu kommen die Aufgaben *relate*, *history* und *extract*. So sollen Beziehungen zwischen den Elementen einsehbar sein und es ist dafür Sorge zu tragen, dass Aktionen rückgängig gemacht und wiederholt werden können. Eine sinnvolle Ergänzung ist außerdem die Möglichkeit, gewonnene (Teil-)Informationen verfügbar und speicherbar zu machen [vgl. Shn03, S. 367-368]. Charakteristisch für die Informationsvisualisierung ist demnach, dass Daten auf verschiedene Weise betrachtet werden können, indem man dafür unterschiedliche Bilder erzeugt. Die visuelle Exploration wird dabei durch Interaktionsmöglichkeiten und algorithmische Berechnungen unterstützt. Diese Merkmale machen die Informationsvisualisierung zu einem interaktiven Geschehen, bei dem visuelle und automatische Methoden in Kombination angewandt werden [vgl. SM04, S. 135].

2.2.2 Visualisierung von Relationen

Eine der Aufgaben dieser Bachelorarbeit besteht darin, eine geeignete visuelle Repräsentation der zu erfassenden Daten im Sinne der Informationsvisualisierung zu finden. Es wird also darum gehen, für die Darstellung von Netzwerken und Gemeinsamkeiten aus den zahlreichen Visualisierungstechniken ein Layout zu wählen, das zum einen die Daten visuell effektiv und

einnehmend abbildet und zum anderen für die Zielgruppe zugänglich und erforschbar macht.

Graphen sind eine wesentliche Darstellungsform von strukturierten und zusammenhängenden Daten. Zunächst sollen daher Terminologie und Eigenschaften dieses mathematischen Modells erläutert werden, um im Anschluss Methoden und Techniken zur Visualisierung von Graphen vorzustellen.

2.2.2.1 Graphen - Terminologie und Eigenschaften

Die Graphentheorie ist zurückzuführen auf den Mathematiker Leonhard Euler (1707-1783) und das *Königsberger Brückenproblem* aus dem 18. Jahrhundert. Die Stadt Königsberg (heute Kaliningrad) liegt am Fluss Pregel, dessen Seitenarme eine Stadtinsel umfliessen und es gibt sieben Brücken, die den Fluss und seine Seitenarme überqueren (siehe Abbildung 2.2). Damals gab es eine Kontroverse zu der Frage, ob es möglich wäre, einen Weg durch die Stadt zu gehen und diese sieben Brücken nur ein einziges Mal zu passieren und wieder am Ausgangspunkt anzukommen. Indem Euler das Problem auf mathematische Elemente wie Linien (als Repräsentation der Brücken) und Punkte (als Repräsentation der Ufer) reduzierte, konnte er herausfinden, dass dies nicht möglich ist, da es Ufer gibt, die mit einer ungeraden Zahl von Brücken verbunden sind [vgl. Gra]. Dieses Ergebnis ist in Abbildung 2.3 zu sehen.

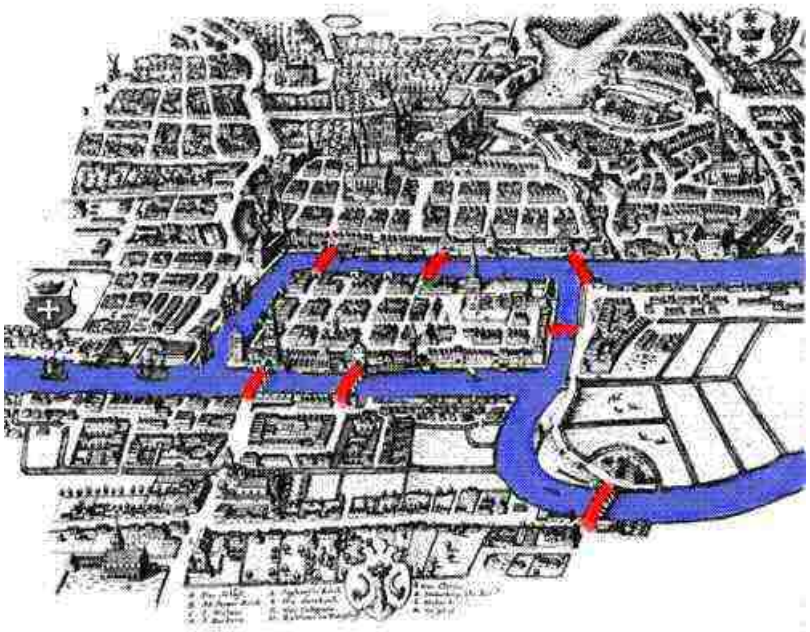


Abbildung 2.2: Die sieben Brücken von Königsberg (Quelle: [Koe]).

Die Schreibweise in Abbildung 2.3 stellt einen Graph dar. Ein Graph wird in der Mathematik symbolisiert durch eine Menge von Punkten, wobei ein Punkt als *Knoten* (engl. *node*) oder

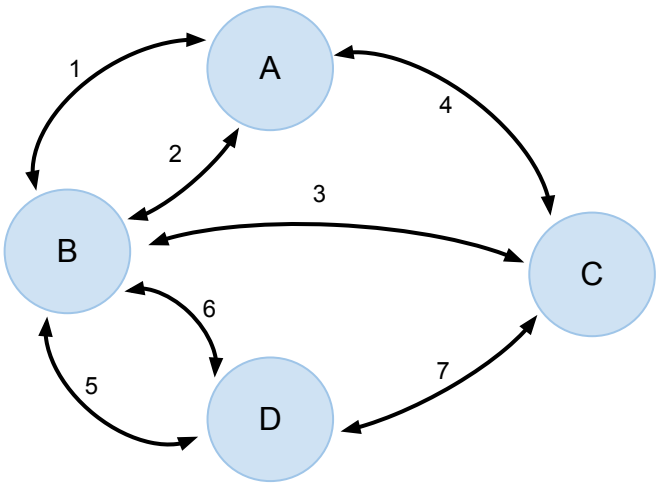


Abbildung 2.3: Das Brückenproblem von Königsberg als Graph dargestellt. Die Punkte A-D definieren die Knoten, die Verbindungslinien 1-7 die Kanten des Graphen.

auch als *Eckpunkt* (engl. *vertex*) bezeichnet wird und einer Menge von Linien, welche die Bezeichnung *Kanten* (engl. *edges*) erhalten. Mit den Kanten werden die Verbindungen (Beziehungen, Relationen) zwischen den Knoten eines Graphen ausgedrückt. Man spricht von einem *gerichteten* Graphen, wenn die Kanten gerichtet sind, d.h. einen Richtungspfeil aufweisen. Handelt es sich um einfache Linien, dann sind die Kanten und der Graph *ungerichtet*. Dies lässt sich wie in Abbilddung 2.3 auch durch beidseitig gerichtete Kanten beschreiben. Ebenfalls zu den Graphen gehören Darstellungen von *Bäumen*. Ein Baum ist dabei ein zusammenhängender, kreisfreier Graph [vgl. Gra].

Mit einer gerichteten Kante lässt sich die einseitige Beziehung eines Knotens zu einem andern darstellen (z.B. Vorgesetzte-Angestellten-Beziehung), hingegen wird eine beidseitige Beziehung durch die Verwendung einer ungerichteten Kante veranschaulicht (z.B. Freundschaft). Graphen eignen sich daher sehr gut für die Visualisierung von Relationen, sowohl von Hierarchien als auch von Netzwerken.

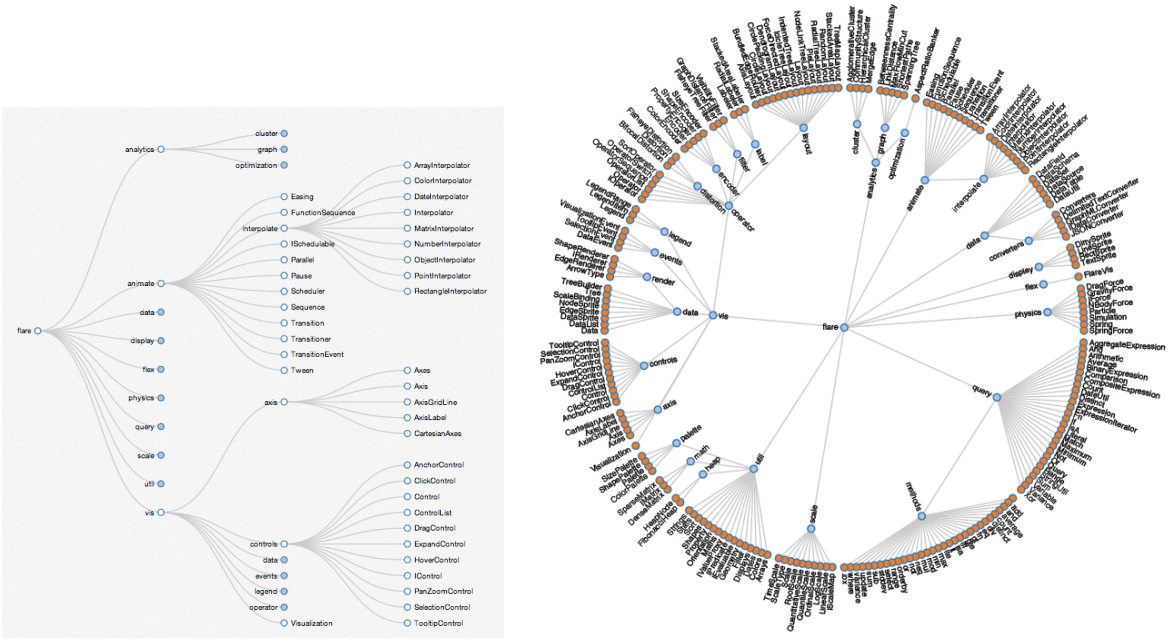
2.2.2.2 Graphvisualisierung

Bei der Graphvisualisierung lassen sich bestimmte Anforderungen an das Aussehen eines Graphen formulieren, die sich an die zugrunde liegenden Layoutalgorithmen richten, beispielsweise in Bezug auf die gleichmäßige Verteilung der Knoten und Kanten, die Kantenlänge oder die Anzahl der Kantenüberschneidungen. Dies sind Faktoren, die mit der praktischen Umsetzung einer Anwendung zusammenhängen (z.B. hinsichtlich des Ausgabebildschirms und dem Umfang des visualisierten Graphen), daneben aber auch die Benutzbarkeit durch die Zielgruppe ansprechen.

Die Daten einer Graphstruktur zu verstehen und einer gründlichen Analyse zu unterziehen, hängt somit unmittelbar vom gewählten Layout ab und die Herausforderung bei der Gestaltung einer visuellen Repräsentation dieser Daten besteht darin, die Dateneigenschaften in geeigneter Form auf visuelle Variablen abzubilden. Die folgenden Beispiele sollen zeigen, dass sich zur Visualisierung von Hierarchien und Netzwerken jeweils unterschiedliche Layouttechniken eignen, um mit visuellen Eigenschaften wie Position, Größe, Form und Farbe die inhärente Struktur der Relationen auszudrücken.

Visualisierung von Hierachien [vgl. BOH10, S. 13-17]

Die Visualisierung von Hierarchien erfolgt häufig in einer Baumstruktur als Knoten-Kanten-Diagramm (engl. *node-link-diagram*). Verwendung findet hier ein *Tree Layout*-Algorithmus, einer der bekanntesten ist dabei der Algorithmus von *Reingold und Tilford* [vgl. HMM00; BOH10, S. 13]. Dieser kann sowohl mit kartesischen Koordinaten als auch mit Polarkoordinaten angewandt werden. Letzteres ist eine ästhetisch reizvolle Variante und nutzt den zur Verfügung stehenden Raum effizient aus. Beide Varianten sind in Abbildung 2.4 dargestellt.



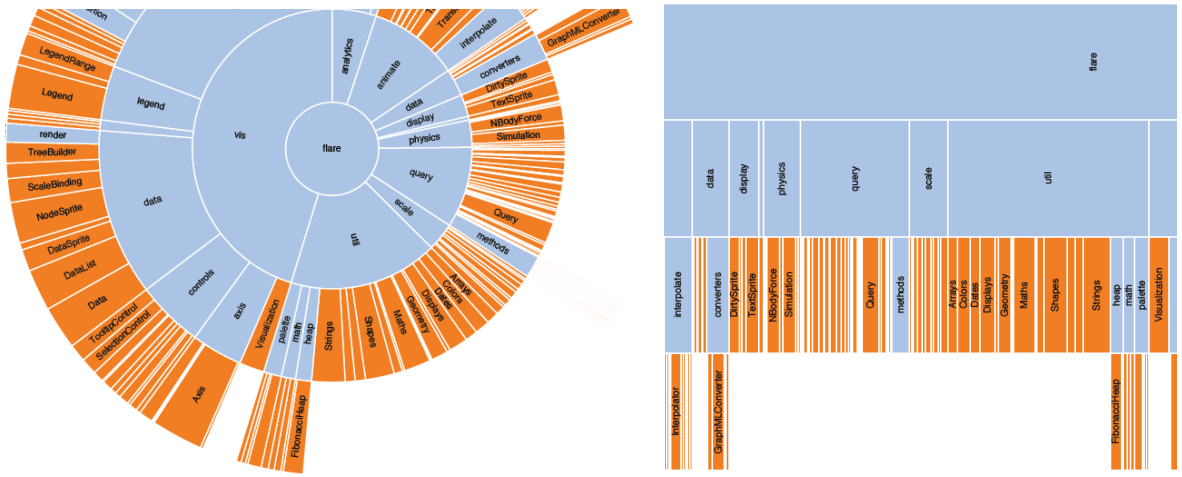
(a) Tree Layout, kartesisch

(b) Tree Layout, polar

Abbildung 2.4: Beispiele für Knoten-Kanten-Diagramme nach dem „Reingold und Tilford"-Algorithmus. Die Anordnung der Hierarchie erfolgt in (a) mit kartesischen Koordinaten von links-nach-rechts, das Layout in (b) verwendet Polarkoordinaten (Quelle: (a) [Tre], (b) [BOH10]).

Das *Icicle Layout* und das *Sunburst Layout* sind Variationen eines Knoten-Kanten-Diagramms,

zu sehen in Abbildung 2.5. Hier wird die Hierarchie jedoch nicht durch eine Kante zwischen Eltern- und Kindknoten ausgedrückt, sondern die Knoten werden als Fläche gezeichnet und die Rangfolge in der Hierarchie erschliesst sich aus der Anordnung relativ zu den adjazenten Knoten. Diese raumfüllenden Diagramme ermöglichen eine weitere visuelle Codierung, da nun über die Länge der flächig gezeichneten Knoten eine zusätzliche Aussage über die Daten gemacht werden kann. Auf ähnliche Weise wird Hierarchie beispielsweise in Abbildung 2.8(a) mit dem *Nested Circle Layout* durch die gebündelte Darstellung und auch über die Größe der Knoten visualisiert.



(a) Sunburst Layout, polar **(b)** Icicle Layout, kartesisch

Abbildung 2.5: Beispiele für raumfüllende Knoten-Kanten-Diagramme zur Darstellung von Hierarchien (Quelle: [BOH10]).

Visualisierung von Netzwerken [vgl. BOH10, S. 17-20]

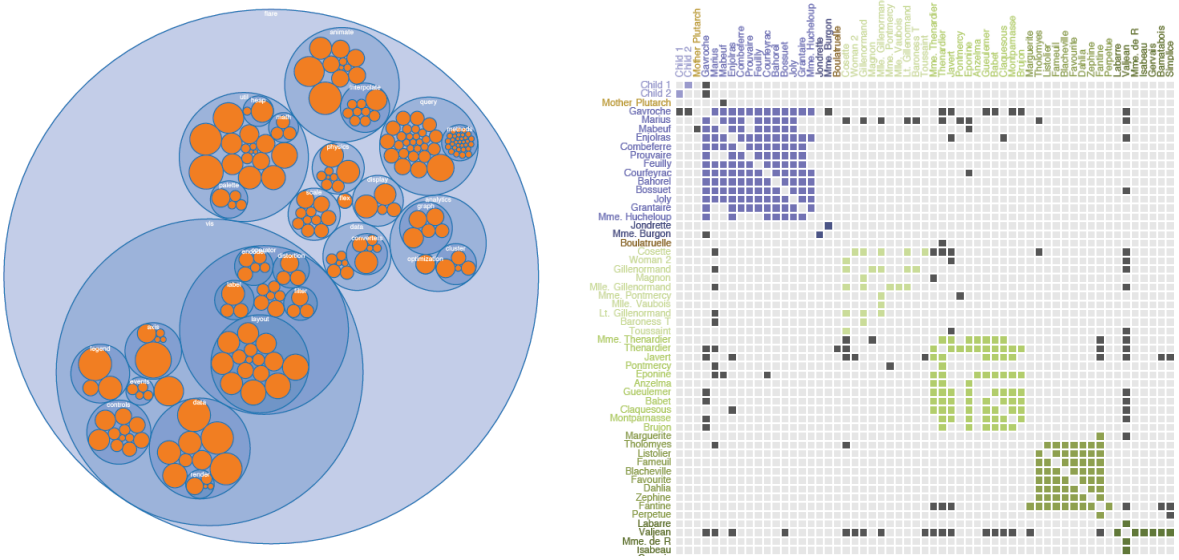
Knoten-Kanten-Diagramme sind ebenfalls die typische Technik zur Visualisierung von Netzwerken. Da jedoch keine Hierarchie abgebildet wird, sind andere Layoutalgorithmen gefragt, welche den oben bereits erwähnten Anforderungen zur Graphvisualisierung gerecht werden. Die Anordnung der Knoten und Kanten oder auch die Distanzen zwischen den Knoten spielen hier eine wichtige Rolle, damit die Gemeinsamkeiten und Beziehungen innerhalb des Netzwerkes zum Ausdruck gebracht werden können.

Force Directed Layout ist die Bezeichnung für eine Visualisierungstechnik, die durch eine Simulation physikalischer Kräfte die Knoten und Kanten auf dem Bildschirm automatisch arrangiert. Als elektrisch aufgeladene Teilchen erscheinen dabei die Knoten, die sich gegenseitig abstoßen. Da die Kanten als Federn zwischen den Knoten fungieren, bleiben diese jedoch

verknüpft und können nicht komplett auseinander treiben. Interaktiv kann dann durch Ziehen an den Knoten das Layout bis zu einem gewissen Grad verändert werden, was wiederum eine ganz eigene spielerische Anziehungskraft auf die Zielgruppe ausüben kann. Ein Beispiel ist in Abbildung 2.6 zu sehen.

Das *Bogendiagramm* (siehe Abbildung 2.7) verwendet zur Netzwerkvisualisierung eine eindimensionale Darstellung der Knoten. Kanten werden in dieser Art von Diagramm zu Kreisbögen, um die Beziehungen zwischen den Knoten zu repräsentieren. Damit dieses Layout ein gutes Ergebnis erzielt, d.h. Strukturen erkannt werden können, ist die Reihenfolge der Knoten ausschlaggebend.

Abbildung 2.8(b) zeigt wie ein Graph auch in Form einer *Adjazenz-Matrix* visualisiert werden kann. Diese Form erfordert ebenso wie das Bogendiagramm eine gute Anordnung der Knoten (hier in Form der Spalten- und Zeileneinträge) und es ist in einer Matrix beispielsweise auch schwierig, einen Pfad auszumachen. Demgegenüber stehen allerdings einige nicht zu verachtende Vorteile, denn es gibt in einer Matrix keine Kantenüberschneidungen und mit einer ordentlichen Knotenreihenfolge und Farbkodierung lassen sich schnell Zusammenhänge identifizieren. Werden überdies der Zielgruppe noch interaktive Möglichkeiten angeboten, z.B. eine Neusortierung der Matrix vorzunehmen, dann können daraus wertvolle Erkenntnisse gewonnen werden.



(a) Nested Circle Layout **(b)** Adjazenz-Matrix

Abbildung 2.8: (a) Hierarchiedarstellung durch Bündelung, (b) Netzwerkvisualisierung in einer Adjazenz-Matrix (Quelle: [BOH10]).

Abbildung 2.6: Force Directed Layout: Die Simulation physikalischer Kräfte bestimmt die automatische Positionierung der Knoten, durch Farbkodierung kann Gruppenzugehörigkeit beschrieben werden (Quelle: [BOH10]).

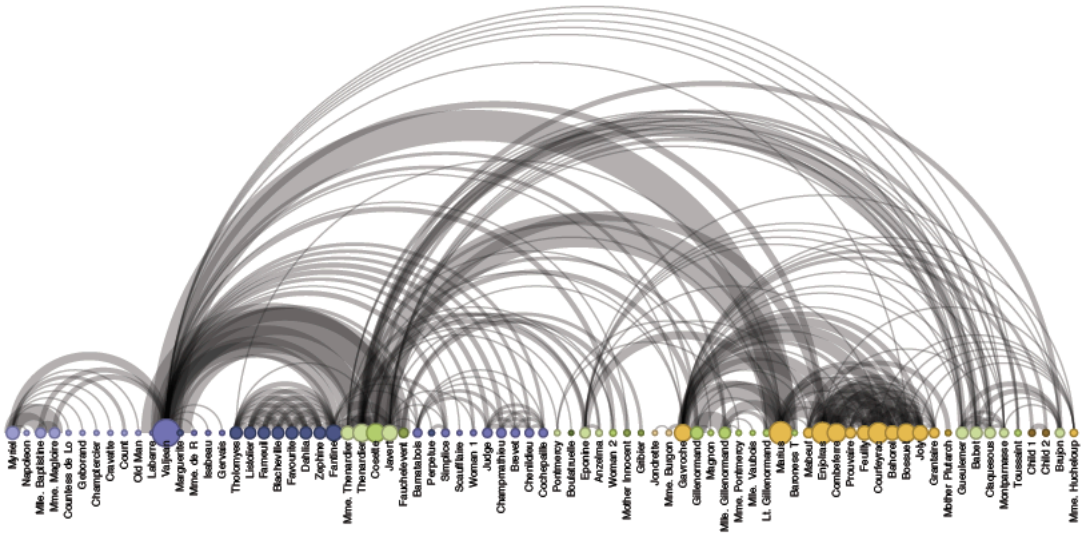
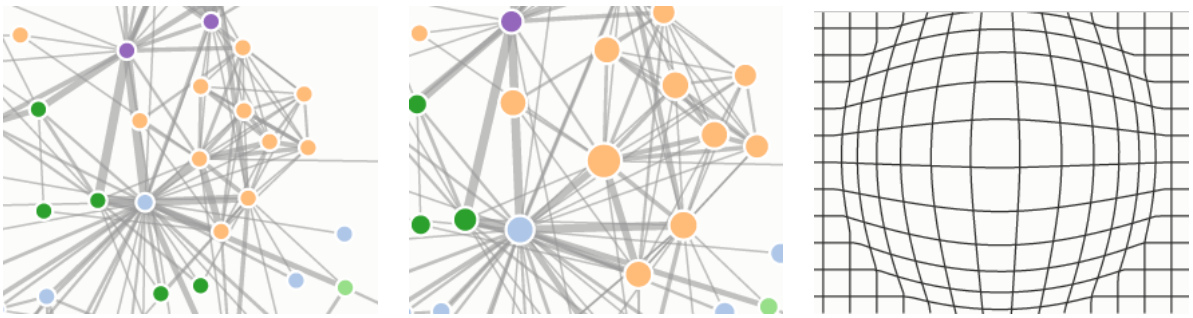


Abbildung 2.7: Das Bogendiagramm zeigt, dass trotz eindimensionaler Darstellung der Knoten dieses Netzwerk-Graphen, Strukturen und Relationen durch die als Kreisbögen gezeichneten Kanten visualisiert werden können (Quelle: [BOH10]).

Navigation und Interaktion [vgl. HMM00]

Informationsvisualisierung bedeutet immer auch Interaktion. Daher sind Werkzeuge zur Navigation und Interaktion unerlässlich. Hierzu gehört z.B. *Zoom & Pan* zum Vergrößern und Verkleinern bzw. Verschieben der Bildschirmanzeige. Neben einer einfachen Vergrößerung des Graphen (*geometric zooming*) ist dabei auch eine inhaltliche Vergrößerung (*semantic zooming*) denkbar, d.h. das detailliertere Informationen sichtbar werden, wenn man in die Visualisierung hineinzoomt. Allerdings entsteht bei der Anwendung von Zooming oft auch ein Kontextverlust, der sich erschwerend auf die Benutzerfreundlichkeit auswirkt. *Focus & Context*-Techniken können jedoch helfen, diese Probleme zu verringern, beispielsweise die sogenannte *Fisheye Distortion*. Dabei wird der aus der Fotografie bekannte Effekt eines Fischaugen-Objektivs nachgeahmt, sodass sich der fokussierte Bereich erweitert und die umliegenden Bildbereiche nur noch verzerrt und damit weniger genau dargestellt werden, ohne dass der Kontext verloren geht.



(a) (b) (c)

Abbildung 2.9: Beispiel für die „Fisheye Distortion"-Technik: (a) zeigt den Ausschnitt eines Graphen bei normaler Ansicht. Die Visualisierung reagiert auf Mouseover, sodass sich der Bereich um die Position der Maus vergrößert, der Kontext des gesamten Graphen aber erhalten bleibt. (b) zeigt den gleichen Ausschnitt bei Mouseover. (c) soll den Verzerrungseffekt anhand eines Gitterrasters noch einmal verdeutlichen (Das Beispiel kann „live" unter der Adresse *http://bost.ocks.org/mike/fisheye/* ausprobiert werden, Quelle: [Fis]).

Die aufgeführten Beispiele stellen sicherlich nur eine Auswahl der vielfältigen Möglichkeiten im Bereich der Informationsvisualisierung und speziell der Verfahren zur Graphvisualisierung dar, sollen aber dazu dienen, einen Eindruck zu vermitteln und Ideen für die Umsetzung der Visualisierung im Rahmen dieser Bachelorarbeit zu gewinnen.

2.3 Technologien

In diesem Abschnitt werden die Technologien vorgestellt, die im Rahmen dieser Bachelorarbeit eine Rolle spielen. Dazu gehören Basistechnologien und Standards wie HTML und JavaScript, die für die Umsetzung einer modernen Web-Applikation unerlässlich sind, hinzu kommen jedoch auch Technologien, die sich im Zuge der Recherche als potentiell relevant zum Gelingen der Aufgabe ergeben haben.

2.3.1 HTML5 und CSS3

Die Aufgabe von HTML geht bereits aus dem Namen *HyperText Markup Language* hervor und ist das Auszeichnen (*mark up*) von Webinhalten, d.h. diese zu beschreiben und in eine logische Struktur zu bringen. Erreicht wird dies durch hierarchische Gliederungen und Beziehungen, die sich durch das HTML-Schema für die verschiedenen Bereiche einer Webseite ausdrücken lassen [vgl. MG10, S. 19-20].

Der derzeitige HTML-Standard ist Version 5, diese darf aber nicht als abgeschlossene Spezifikation verstanden werden. Vielmehr befindet sich HTML5 seit 2007 stetig in der Entwicklung [vgl. MG10, S. 20]. Daran beteiligt sind das *World Wide Web Consortium* (*W3C*)[2] und die *Web Hypertext Application Technology Working Group* (*WHATWG*)[3]. Letztere befasst sich vor allem mit den technologischen und praktischen Aspekten der Webstandards und besteht zum großen Teil aus Interessensvertretern der Webbrowser-Anbieter, das W3-Konsortium ist hauptsächlich für die Dokumentation und die Beschreibung der Spezifikation zuständig [vgl. MG10, S. 39-40].

Mit HTML5 wurden neue Sprachelemente eingeführt. Diese lassen sich unterscheiden in Elemente, die der Textstrukturierung dienen und eine sinnvolle Bedeutung ausdrücken, an die jedoch keine Funktionalität gebunden ist. Hierzu zählt z.B. das `article`-Element, um Blogartikel oder Forenbeiträge auszuzeichnen. Anderen Elementen kommt hingegen eine Funktion zu, die der Webbrowser unterstützen muss. Dazu gehören u.a. die Elemente `video`, `audio` oder `canvas` zur Wiedergabe multimedialer Inhalte bzw. zur Darstellung von 2D- und 3D-Grafiken [vgl. MG10, S. 132].[4] Zur grafischen Gestaltung bestimmter Bereiche einer Webseite gibt es neben dem `canvas`-Element seit HTML5 auch die Möglichkeit eine SVG-

[2] vgl. *http://www.w3.org/Consortium/* (besucht am 30.08.2013)
[3] vgl. *http://www.whatwg.org/* (besucht am 30.08.2013)
[4] Um zu prüfen, ob ein Webbrowser die nötige Funktionsunterstützung erbringt, gibt es mittlerweile viele im World Wide Web verfügbaren Übersichten. Ein Beispiel für eine solche Webbrowser-Kompatibilitätstabelle ist die Webseite „Can I use...“ unter der Adresse *http://caniuse.com/* (besucht am 30.08.2013).

Grafik direkt in ein HTML-Dokument einzubinden. SVG steht für *Scalable Vector Graphics* und ist ein text-basiertes Bildformat, das durch eine ähnliche Auszeichnungssprache wie HTML definiert wird [vgl. Mur13, S. 50]. SVG wird von der Visualisierungsbibliothek D3 genutzt und somit bei der Umsetzung der Web-Applikation eine Rolle spielen (siehe Abschnitt 2.3.2.5).

Die oben erwähnte hierarchische Struktur eines HTML-Dokuments und seiner Elemente wird bei der Verarbeitung durch einen Webbrowser intern als Baumstruktur dargestellt. Diese Struktur wird als *Document Object Model* (DOM) bezeichnet und definiert die Schnittstelle, die es Programmiersprachen erlaubt, auf die Elemente des Dokuments zuzugreifen und diese zu manipulieren sowie neue hinzuzufügen (in Zusammenhang mit JavaScript als *DOM Scripting* bezeichnet) [vgl. MG10, S. 347-348]. Das DOM ist elementar für die Realisierung dynamischer, interaktiver Webseiten und Web-Applikationen sowie webbasierter Visualisierungen. Die in den modernen Webbrowsern integrierten Entwicklungswerkzeuge (z.B. die *Developer Tools* in Chrome) ermöglichen die Sicht auf den aktuellen Zustand des DOM. Dies ist während der Entwicklung sehr nützlich, da man zum einen die Struktur eines Dokuments erkunden und ein besseres Verständnis dafür entwickeln kann und zum anderen hilft dies bei der Fehlersuche und -beseitigung.

Ergänzend zur Auszeichnungssprache HTML werden *Cascading Style Sheets* eingesetzt, um den Inhalt einer Webseite zu formatieren und visuell zu gestalten. Cascading Style Sheets funktionieren über Regeln, die für die zu gestaltenden HTML-Elemente formuliert werden, also z.B. Regeln für Größe, Farbe, Position oder Schriftart. Die Standardisierung erfolgt ebenfalls über das W3-Konsortium und auch bei CSS ist die aktuelle Version 3 in ständiger Entwicklung. Die Spezifikation ist bei CSS3 in einzelne Module getrennt (z.B. *CSS Animations* oder *CSS Backgrounds and Borders*), die sich jeweils in eigenen Entwicklungsstadien befinden.[5]

Die Web-Applikation soll mithilfe des Front-End/CSS-Framework Bootstrap von Twitter (*http://getbootstrap.com/2.3.2/*)[6] entwickelt werden, um auf bereits vorhandene Gestaltungsvorlagen mit vordefinierten CSS-Regeln zurückgreifen zu können. Front-End/CSS-Frameworks sind zu einem festen Bestandteil in der Welt der Webentwicklung geworden, da sie erforderliche Arbeitsschritte beschleunigen, insbesondere wenn es um die Entwicklung eines ersten Prototyps geht. Zusätzlich unterstützen die Frameworks dabei, Benutzeroberflächen so zu gestalten, dass diese sowohl auf einem Desktop-PC als auch auf einem mobilen Gerät sowie in den unterschiedlichen Webbrowsern ansprechend aussehen und vor allem bedienbar bleiben.

[5] vgl. *http://www.w3.org/Style/CSS/current-work* (besucht am 30.08.2013)
[6] (besucht am 30.08.2013)

2.3.2 JavaScript

JavaScript wurde Mitte der 90er Jahre von dem Unternehmen *Netscape* als browserseitige Skriptsprache für ihren Webbrowser *Netscape Navigator 2.0* entwickelt und ist seitdem aus der Welt der Webbrowser nicht mehr wegzudenken. Tatsächlich ist JavaScript sogar die einzige Sprache, die heute von allen Webbrowsern unterstützt wird, da andere Ansätze wie Java- oder Flash-Plugins sich nicht durchsetzen konnten. Die anfänglichen Aufgaben wie das simple Beeinflussen einzelner HTML-Elemente einer Webseite sind mittlerweile um ein vielfaches erweitert worden, sodass inzwischen komplexe Webseiten und vollständige Web-Applikationen mit JavaScript programmiert werden. Da das World Wide Web in den letzten Jahren eine wichtige Rolle als Plattform für die Anwendungsentwicklung eingenommen hat, gewinnt die Entwicklung von Web-Applikationen zunehmend an Bedeutung. Der offizielle Sprachstandard von JavaScript wird unter der Bezeichnung *ECMAScript*[7] von der Organisation *Ecma International* (früher: *European Computer Manufacturers Association*)[8] spezifiert und ist seit Dezember 2009 als Version 5 anerkannt [vgl. Ste10, S. 1; Cro08, S. 4].

JavaScript ist eine objektorientierte Programmiersprache, besitzt aber kein Klassensystem für die Erzeugung von Objekten. Stattdessen können Objekte dynamisch bei Bedarf erzeugt und Eigenschaften hinzugefügt werden. Eine Vererbung von Eigenschaften ist aber dennoch möglich, indem ein Objekt die Eigenschaften von anderen Objekten erben kann. Dieses Vererbungsprinzip nennt man *prototypische Vererbung*. In JavaScript stellt alles, mit Ausnahme der primitiven Datentypen `number`, `string`, `boolean`, `null` und `undefined`, ein Objekt dar. Das gilt auch für Funktionen und als Objekte können Funktionen in Variablen, Arrays oder anderen Objekten gespeichert werden sowie Eigenschaften und Methoden besitzen [vgl. Ste10, S. 3-4; Cro08, S. 26]. Dies macht Funktionen zu sogenannten “first-class objects“ [Ste10, S. 1], die vielfältige Aufgaben in JavaScript übernehmen.

Eine weitere Eigenschaft von JavaScript ist die schwache Typisierung, d.h. der Datentyp einer Variablen muss bei der Deklaration nicht angegeben werden, sondern die Deklaration erfolgt lediglich mit der Anweisung `var` und JavaScript ordnet den Datentyp dann automatisch zu, je nachdem welche Art von Information der Variablen zugewiesen wird, dies ist in Zeile 1 bis 3 von Listing 2.1 dargestellt. Die schwache Typisierung hat zur Folge, dass man eine Variable leicht überschreiben kann, ohne dass ein Fehler auftritt (Zeile 5-7).

Listing 2.1: Schwache Typisierung in JavaScript

```
1 var x = 4; // number
```

[7] vgl. *http://www.ecmascript.org/* (besucht am 28.08.2013)
[8] vgl. *http://www.ecma-international.org/memento/history.htm* (besucht am 28.08.2013)

```
2 var y = "Hello World"; // string
3 var z = true; // boolean
4
5 var myVar = "4"; // string
6 myVar = 19; // number
7 myVar = true; // boolean
```

Die Angabe von `var` ist nicht strikt erforderlich, d.h. es ist sogar möglich, eine Variable ohne vorherige Deklaration zu benutzen. Davon ist jedoch abzuraten, da man dadurch eine globale Variable erzeugt, auf die von überall zu jeder Zeit zugegriffen werden kann. Seit ECMAScript 5 gibt es daher einen sogenannten *strict mode*, den man aktivieren kann und der bei Verwendung einer Variable ohne `var`-Anweisung einen Fehler erzeugt [vgl. Ste10, S. 5].

Die „natürliche“ Umgebung eines JavaScript-Programms ist der Webbrowser. Auch dort befinden sich Objekte. Das oberste Objekt in dieser Umgebung ist das `window`-Objekt und definiert den *globalen Namensraum* (engl. *global namespace*) und damit den Geltungsbereich der Umgebung (*global scope*). Eine Variable außerhalb einer Funktion zu deklarieren, ist eine weitere Möglichkeit, globale Variablen zu erzeugen. Deshalb ist es wichtig, bei der Programmierung einer JavaScript Web-Applikation darauf zu achten, nicht zu viele globale Variablen dem globalen Namensraum hinzuzufügen, denn mit dem Einbinden von immer mehr JavaScript-Bibliotheken in eigene Anwendungen erhöht sich die Gefahr von Namenskonflikten mit anderen globalen Variablen, was ein Programm fehleranfällig macht und zu unerwartetem Verhalten führen kann. Im Rahmen dieser Bachelorarbeit soll es daher auch darum gehen, bei der Programmierung die Konventionen einzuhalten, Variablen immer mit der `var`-Anweisung und innerhalb von Funktionen zu deklarieren sowie weitere geeignete *Coding Patterns* zu finden und Werkzeuge einzusetzen, die bei der Umsetzung der Web-Applikation unterstützen und helfen, Fehler zu vermeiden.

Die nachfolgenden Abschnitte widmen sich zum einen typischen Technologien, welche die Entwicklung einer JavaScript Web-Applikation begleiten und zum anderen sollen die bereits ausgewählten JavaScript-Hilfsmittel vorgestellt werden.

2.3.2.1 JSON

Das Kurzwort JSON (*http://json.org/*)[9] steht für *JavaScript Object Notation* und ist ein Datenaustauschformat. JSON baut auf Datenstrukturen auf, die von den meisten Programmiersprachen unterstützt werden oder die dort in ähnlicher Form vorhanden sind. Dazu gehören

[9](besucht am 29.08.2013)

beispielsweise Objekte und Arrays [vgl. Cro08, S. 136]. Da JSON aus der JavaScript-Welt stammt, lässt sich das Format schneller von JavaScript verarbeiten als beispielsweise *XML (Extensible Markup Language)*, was ein alternatives Format zum Datenaustausch ist. JSON eignet sich daher gut für die Verwendung mit *Ajax* (siehe 2.3.2.2) und der Kommunikation mit einem Web- oder Datenbankserver [vgl. Mur13, S. 39].

2.3.2.2 Ajax

Asynchronous JavaScript and XML oder abgekürzt Ajax ist eine Bezeichnung für das Zusammenspiel verschiedener Technologien, das eine asynchrone Kommunikation zwischen Webbrowser und Webserver möglich macht. Die Hauptrolle spielt hier das *XMLHttpRequest*-Objekt, ein spezielles JavaScript-Objekt, mit dessen Hilfe JavaScript in der Lage ist, während eine Webseite angezeigt wird, im Hintergrund beim Webserver Daten anzufordern, die sich dann in das bereits angezeigte HTML-Dokument einbauen lassen. Asynchron bezieht sich demnach darauf, dass lediglich neue Daten nachgeladen werden und kein komplett neues Dokument. Die Rückmeldung ist dadurch viel schneller und flüssiger, was die Realisierung interaktiver Anwendungen ermöglicht. Ajax hat wesentlich zu dem Aufkommen von Web-Applikationen beigetragen, die sich wie lokal installierte Programme bedienen lassen [vgl. MG10, S. 24, 360-361].

2.3.2.3 jQuery

JavaScript ist in den einzelnen Webbrowsern zum Teil unterschiedlich implementiert, d.h. eine Web-Applikation kann in Chrome einwandfrei funktionieren, reagiert jedoch nicht wie erwünscht im Internet Explorer. Die meisten aller wichtigen und häufig erforderlichen Aufgaben bei der Programmierung von Web-Applikationen lassen sich daher besser und auf einfachere Weise mit dem Einsatz sogenannter *JavaScript-Bibliotheken* bewältigen, da man auf eine Sammlung fertiger JavaScript-Funktionen zurückgreifen kann, die bereits auf mögliche Webbrowser-Unterschiede getestet sind und dadurch das Fehlerpotential sowie den Programmieraufwand verringern [vgl. SM11, S. 117-118].

jQuery (*http://jquery.com/*)[10] ist eine bekannte und beliebte JavaScript-Bibliothek und stellt zudem eine umfangreiche Dokumentation mit Anwendungsbeispielen bereit. Insbesondere die Bearbeitung und der Zugriff auf das Document Object Model (DOM) von HTML-Dokumenten wird erleichtert, da jQuery mit CSS-Selektoren arbeitet und das Verketten von Funktionen erlaubt, wodurch umfangreichere Zugriffe auf die Dokumentknoten möglich sind. Ebenfalls

[10](besucht am 29.08.2013)

wird mittels jQuery das Senden von *HTTP-Requests*[11] an einen Web- oder Datenbankserver vereinfacht, da vollständige Funktionen zur Nutzung der Ajax-Technologie (vgl. 2.3.2.2) genutzt werden können.

2.3.2.4 Handlebars

Bei der Entwicklung von clientseitigen JavaScript Web-Applikationen mit dynamisch generierten Inhalten helfen *Template Engines* dabei, zwischen HTML und JavaScript-Quellcode zu trennen. Die Template Engine Handlebars (*http://handlebarsjs.com/*)[12] soll daher bei der Umsetzung des Prototyps eingesetzt werden.

Eine Template Engine kann als eine JavaScript-Bibliothek zur Verarbeitung von HTML-Vorlagen, sogenannten *Templates*, verstanden werden. Ein Handlebars-Template besteht aus gewöhnlichem HTML mit darin definierten Bereichen, die durch Handlebars-Ausdrücke gekennzeichnet sind. Bei der Verarbeitung durch die Template Engine werden diese mit den geladenen Inhalten ersetzt und das HTML-Ergebnis wird mit den eingefügten Werten als String zurückgegeben und dem HTML-Dokument hinzugefügt [vgl. Hbr].

Für den Einsatz von Handlebars sind die folgenden drei Hauptbestandteile wichtig:

1. Der Handlebars-Ausdruck, bestehend aus einem Paar doppelt geschweifter Klammern, die eine Variable oder auch eine Funktion umschließen: `{{variable}}`.
2. Die `Handlebars.compile()`-Funktion, zur Verarbeitung von HTML und Handlebars-Ausdrücken.
3. Der `context`, d.h. der Dateninhalt. Dieser wird als JavaScript-Objekt übergeben. Enthält das Datenobjekt ein Array mit Objekten, kann der Block-Ausdruck `{{#each}}` verwendet werden.

Neben dem schon genannten `{{#each}}` Block-Ausdruck, gibt es noch einige weitere sogenannte *Block Helper* für Schleifen und Bedingungen, z.B. `{{#if}}` oder `{{#unless}}` und Handlebars erlaubt es auch, eigene *Custom Helper*-Funktionen zu schreiben. Damit können partiell Kontrollstrukturen auf das HTML-Template angewendet werden, Handlebars gilt dennoch als Template Engine, die „logic-less“ [Hbr] ist und es erlaubt, die HTML-Templates einfach und von den logikbasierten JavaScript-Dateien entkoppelt zu halten.

[11] HTTP steht für *Hypertext Transfer Protocol* und definiert das Protokoll zur Übertragung von Daten im World Wide Web zwischen Webbrowser und Webserver. Mit einem HTTP-Request fordert ein Webbrowser Daten an, die der Webserver in einer HTTP-Response überträgt.

[12] (besucht am 27.08.2013)

2.3.2.5 D3

Das erste Framework zur Visualisierung von Daten im World Wide Web war das *prefuse – Information Visualization Toolkit*[13]. Das in Java geschriebene Visualisierungsframework gibt es seit 2005 und bietet seitdem die Möglichkeit, webbasierte Visualisierungen zu erstellen und via Java-Plugins in Webseiten zu integrieren. Jeff Heer, einer der Autoren von prefuse, veröffentlichte 2007 mit *Flare*[14] ein ähnliches Visualisierungswerkzeug für die Programmiersprache ActionScript. Die mit Flare realisierten Visualisierungen können somit über den Adobe Flash Player in Webseiten bereitgestellt werden. Im Zuge der fortschreitenden Entwicklung im Bereich der Webbrowser-Technologien entstanden jedoch in den letzten Jahren vermehrt Visualisierungsbibliotheken, die auf JavaScript basieren und sich nativ in den Webbrowser integrieren lassen und es dadurch möglich machen, auf Plugins zu verzichten [vgl. Mur13, S. 9-10].

Eine dieser JavaScript-Visualisierungsbibliotheken ist D3 (*http://d3js.org/*)[15]. Der Name D3 ist eine Abkürzung für die volle Bezeichnung *Data-Driven Documents* und ein Hinweis auf die Funktionsweise der Javascript-Bibliothek bzw. die Ergebnisse, die produziert werden können. *Data* bezieht sich demnach auf die Daten und Informationen, die man zur Verarbeitung in einem *Document* bereitstellt. Letzteres adressiert webbasierte Dokumente, also Formate wie HTML und SVG und deren Elemente, die von einem Webbrowser gerendert werden können. D3 ist dabei der *driver*, indem die JavaScript-Bibliothek die Verknüpfung zwischen den Daten und dem Dokument herstellt [vgl. Mur13, S. 7]. D3 wurde 2011 veröffentlicht und Jeff Heer ist abermals als Mit-Autor zu nennen, neben Vadim Ogievetsky und Michael Bostock und alle drei Autoren gehören dem Computer Science Department an der Stanford University in Kalifornien an. D3 vereint in sich alle Komponenten moderner Webtechnologien wie HTML für den Seiteninhalt, CSS für das Layout, JavaScript für die Interaktion und SVG für das Rendern von Vektorgrafiken und verzichtet damit bewusst darauf, lediglich eine abstrakte Repräsentation der generierten Visualisierung abzubilden, da auf die zugrundeliegende Struktur in Form des Document Object Model (DOM) direkt zugegriffen werden kann [vgl. BOH11].

D3 ist als Open Source Projekt frei verfügbar (*https://github.com/mbostock/d3/*)[15] und es steht eine ausführliche Dokumentation zur Verfügung. D3 kann JSON als Datenformat verarbeiten und stellt vordefinierte Layout-Methoden bereit, die bei der Realisierung bestimmter Visualisierungstechniken unterstützen, z.B. um Graphen oder Netzwerke zu visualisieren. Dies sowie

[13] vgl. *http://prefuse.org/* (besucht am 01.09.2013)
[14] vgl. *http://flare.prefuse.org//* (besucht am 01.09.2013)
[15] (besucht am 01.09.2013)

die vielen faszinierenden Beispiele und Online-Tutorials[16] haben nicht zuletzt den Ausschlag gegeben, sich für diese Visualisierungsbibliothek zu entscheiden, um die Aufgabe hinsichtlich der Visualisierung innerhalb der Web-Applikation zu lösen.

2.3.3 Neo4j

Graphen eignen sich nicht nur wie in Abschnitt 2.2.2 erläutert zur Visualisierung von Relationen und Netzwerken, sondern auch zur Verarbeitung von zusammenhängenden und vernetzten Daten. Für die Aufgabe dieser Bachelorarbeit bietet es sich daher an, als Datenbanksystem eine *Graphdatenbank* zu verwenden.

Graphdatenbanken gehören zu den *NoSQL*-Datenbanksystemen, die im Verlauf des letzten Jahrzehnts großen Zuspruch gewonnen haben und eine Alternative zu den klassischen *relationalen* Datenbanksystemen (z.B. von Oracle, IBM oder MySQL) darstellen. So ist auch der Begriff NoSQL als *„Not only SQL“* zu verstehen [vgl. Edl+11, S. 5]. SQL steht für *„structured query language“* und ist die Abfragesprache für relationale Datenbanken. Die Abfrage richtet sich dabei an die Tabellen der relationalen Datenbank, in denen die Daten nach einem strikten Schema gespeichert werden. Der Begriff „relational“ bezieht sich demnach auf die Verknüpfung der Tabellen. Diese Art der Datenspeicherung hat jedoch Grenzen und ist nicht immer optimal. Und so bieten NoSQL-Datenbanken beispielsweise neue Ansätze, wenn es darum geht, spezifische Probleme wie Routenberechnungen zu lösen, mit den enormen Datenmengen des *Web 2.0*[17] umzugehen oder durch ein schemafreies System auf Änderungen zu reagieren, indem sich beispielsweise die Anzahl der Felder für die Erfassung der Daten dynamischer anpassen und erweitern lässt [vgl. Edl+11, S. 2-3].

Eine vollständige Erläuterung der Unterschiede zwischen relationalen und nichtrelationalen Datenbanksystemen sowie der unterschiedlichen NoSQL-Datenbanken kann an dieser Stelle nicht erfolgen, der Vollständigkeit halber sollen letztere aber angeführt werden. Neben Graphdatenbanken sind daher noch *Column Stores*, *Document Stores*, *Key/Value Stores*, *Objektdatenbanken*, *XML-Datenbanken* und *Grid-Datenbanken* als NoSQL-Datenbanksysteme zu nennen [vgl. Edl+11, S. 6].

[16] Die auf der D3-Projektseite verlinkten Beispiele (*https://github.com/mbostock/d3/wiki/Gallery*) sind jeweils ergänzt um einen Quellcodeauszug, daneben gibt es einen eigenen Bereich mit D3-Tutorials unter *https://github.com/mbostock/d3/wiki/Tutorials* (besucht am 01.09.2013).

[17] Der Begriff Web 2.0 subsumiert unter sich eine Vielzahl verschiedener Merkmale und Praktiken, die den Umgang mit dem World Wide Web und dessen Bedeutung seit über 10 Jahren prägen, u.a. das Aufkommen von Blogs, Tauschbörsen oder Sozialen Netzwerken und damit einhergehend auch ein Anstieg der Daten, die gespeichert werden müssen [vgl. O'R05].

Die Graphdatenbank Neo4j (*http://www.neo4j.org*)[18] wird seit 2007 von der Firma Neo Technology entwickelt und gehört zu den bekanntesten Vertretern dieser Art von Datenbanksystemen [vgl. Edl+11, S. 290-291]. Die Entscheidung in dieser Arbeit Neo4j als Datenbankkomponente einzusetzen, beruht neben der spezifischen Eignung von Graphdatenbanken für verknüpfte Daten im wesentlichen noch auf den zwei folgenden Punkten. Eine der von Neo4j zur Verfügung gestellten Programmierschnittstellen (engl. *Application Programming Interface, API*), die Neo4j REST API, kann genutzt werden, um Neo4j als Datenbankserver in ein Projekt zu integrieren. Auf diese Weise kommuniziert der Webbrowser mit dem Datenbankserver über HTTP, was eine vollständige Entwicklung des Prototyps in JavaScript ermöglicht (eine Erklärung zu REST folgt in Abschnitt 4.1.2). Darüber hinaus besticht Neo4j durch eine sehr gute Dokumentation, die zahlreichen, frei verfügbaren Online-Tutorials und Videos zu den verschiedensten Themen rund um Neo4j und Graphdatenbanken im allgemeinen sowie durch die Aktivitäten des Neo4j-Teams in diversen Open Source Communities, was für die Einarbeitung in diese Thematik und Technologie eine wertvolle Lernhilfe und Unterstützung darstellt.

Neben der Neo4j REST API hat das Neo4j-Team eine *Java API* und *Spring Data Neo4j*, eine Bibliothek zur Integration von Neo4j in das *Spring Data Framework*[19], entwickelt. Zahlreiche weitere Schnittstellen und Bibliotheken für andere Programmiersprachen werden von Entwicklern aus der Open Source Community beigesteuert - dies zeugt auch von Neo4js Popularität. Eine Übersicht der aktuellen Anbindungen findet man auf der Neo4j Website unter *http://www.neo4j.org/develop/drivers*.[18]

2.3.3.1 Das Poperty-Graph-Modell

In einer Graphdatenbank werden die Daten in einem Graph gespeichert, d.h in einer Struktur aus Knoten und Kanten (siehe Abschnitt 2.2.2.1). Die englische Neo4j-Terminologie benennt eine Kante jedoch nicht mit dem englischen Begriff *edge* sondern bezeichnet diese als *relationship* (dt. *Beziehung, Relation*). Kanten haben immer eine Richtung und einen sogenannten *relationship type*. Damit ist kein Datentyp gemeint, sondern vielmehr eine Art *Kantenlabel*, das die Kante bezeichnet und identifizierbar macht sowie die Abfrage und Traversierung des Graphen unterstützt. Gleichzeitig wird darüber auch Struktur und Bedeutung ausgedrückt, also die Relationen zwischen durch Kanten verbundene Knoten. Sowohl Knoten als auch Kanten können *Eigenschaften* (engl. *properties*) besitzen, bestehend aus einem *key-value pair*. Dabei ist der *key* (dt. *Schlüssel*) vom Datentyp `String`, für *values* (dt. *Werte*) gelten alle primitiven Datentypen

[18] (besucht am 12.08.2013)
[19] Informationen hierzu siehe *http://www.springsource.org/spring-data/neo4j* (besucht am 07.09.2013).

oder Arrays eines primitiven Datentyps, z.B. `name:"Bob"` oder `married:true`. Ein Graph, der sich durch die genannten Merkmale auszeichnet, ist ein *Property-Graph* und beschreibt das von Graphdatenbanken wie Neo4j verwendete Datenmodell - das *Property-Graph-Modell* [vgl. neo13, S. 11-17; RWE13, S. 26; Edl+11, S. 210-211].

Das Beispiel in Abbildung 2.10 veranschaulicht ein simples Property-Graph-Modell und dort ist erkennbar, dass alle Knoten Eigenschaften besitzen. Knoten, die Personen repräsentieren, verfügen hier über die Eigenschaften `name`, `married` und `age`. Die Eigenschaften `movie` und `genre` hat dagegen ein Knoten, der Informationen über einen Kinofilm speichert. An den vorhandenen Kanten, deren Richtung sowie Bezeichnung lässt sich ablesen, welche Personen sich kennen und welche Personen den Kinofilm gesehen haben. Daraus ergibt sich bereits eine mögliche Datenbankabfrage. Beispielsweise könnte Bob daran interessiert sein, welche Person aus seinem Freundeskreis den Kinofilm noch nicht gesehen hat, damit er ihn empfehlen kann. Denkbar wäre auch, der Kante mit dem Label `HAS_SEEN` eine Eigenschaft `rating` zu geben, sodass sich darin speichern lässt, wie der Kinofilm von den einzelnen Personen bewertet wurde. Dies würde wiederum Tom erlauben, sich anhand der Bewertungen seines Freundes zu informieren, ob es sich lohnt, den Kinofilm anzuschauen.

Interessant ist nun die Frage, wie man in der Neo4j Graphdatenbank Datenoperation durchführen kann. Daher wird im nächsten Abschnitt Cypher vorgestellt, die Neo4j-eigene Graphabfragesprache, die im Rahmen dieser Bachelorarbeit in Verbindung mit Neo4j verwendet werden soll.

2.3.3.2 Cypher

Cypher ist eine *deklarative* Abfragesprache (engl. *declarative query language*) für Graphdatenbanken und speziell für Neo4j entwickelt worden. Deklarativ bedeutet, dass man in einer Anfrage an die Datenbank beschreibt, *was* einen interessiert und nicht, *wie* die Operation in der Datenbank durchgeführt werden soll. So kann das Prinzip, nach dem Cypher funktioniert, als „*pattern matching*“ (dt. *Musterabgleich*) verstanden werden. Ähnlichkeiten mit anderen Abfragesprachen wie SQL bestehen unverkennbar in Schlüsselworten wie `WHERE` oder `ORDER BY` [vgl. neo13, S. 109].

Um Graphdaten zu lesen, zu bearbeiten, neu zu erzeugen oder zu löschen, stellt Cypher verschiedene Anweisungen (engl. *clause*) bereit. Die wichtigsten sind in Tabelle 2.1 aufgeführt [vgl. neo13, S. 109].

Die Tabelle präsentiert lediglich eine Auswahl der Abfragesprache, denn Cypher umfasst noch eine Vielzahl anderer Anweisungen und Ausdrücke sowie Funktionen, Vergleichs- oder Boole-

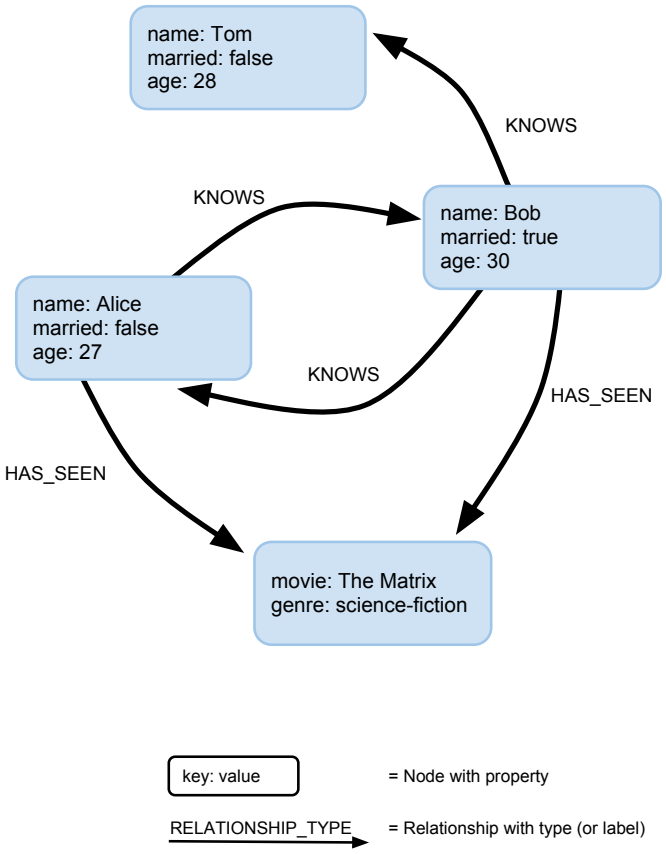


Abbildung 2.10: Beispiel eines Property-Graph-Modell

sche Operatoren, die eine Abfrage komplexer Graphmuster und Datenoperationen ermöglichen. In den Abschnitten 4.1.2 und 4.1.3 wird Cypher noch einmal aufgegriffen, an dieser Stelle soll daher nur ein kurzes Beispiel angeführt werden. Bezogen auf den Property-Graph in Abbildung 2.10 kann man umgangssprachlich zunächst ein Interesse wie folgt formulieren: „Wer ist mit Bob befreundet?“. Für Cypher bedeutet dies die Aufgabe, alle von dem *Bob*-Knoten ausgehenden Relationen mit dem Label `KNOWS` zu finden. Die entsprechende Cypher-Abfrage lautet dann:

```
START bob=node:node_auto_index(name="Bob")
MATCH (bob)-[:KNOWS]->(friend)
RETURN friend.name
```

Mit der `START`-Anweisung wird definiert, dass die Graphtraversierung an dem Knoten mit der Eigenschaft `name="Bob"` starten soll und dass `bob` als sogenannter *identifier* (dt. *Bezeichner*) für den Start-Knoten fungiert. Dadurch kann in der `MATCH`-Anweisung das Muster beschrieben

Tabelle 2.1: Cypher-Anweisungen (Auswahl)

Anweisung	Bedeutung
`START`	Definiert den oder die Start-Knoten.
`MATCH`	Beschreibt das Muster, das erkannt werden soll.
`RETURN`	Definiert die Rückgabe.
`WHERE`	Filtert das Ergebnis.
`CREATE`	Erzeugt Knoten oder Relationen.
`DELETE`	Löscht Knoten, Relationen und Eigenschaften.
`SET`	Setzt die Werte der Eigenschaften.

werden, das von Interesse ist. `friend` dient ebenfalls als Bezeichner für die gefundenen Freundes-Knoten und so können in der `RETURN`-Anweisung die Namen der gefundenen Freunde zurückgegeben werden.

Das besondere an Cypher ist sicherlich auch die Syntax. Dahinter steckt die Philosophie, eine verständliche und einfach zu lesende Sprache zu entwickeln, mit der es intuitiv möglich ist, ein Graphmuster zu beschreiben, z.B. zwei Knoten `a` und `b`, die mit einer Relation `r` verbunden sind: `(a)-[r]->(b)`. Die Art und Weise, wie durch die Verwendung von Sonderzeichen die stilisierte Darstellung von Knoten und Relationen erreicht wird, überträgt gewissermaßen die „natürliche“ Gestalt eines Graphen auf die Ebene der Datenbank [vgl. RWE13, S. 27-28].

Kapitel 3

Konzeption und Entwurf der Applikation

Im Mittelpunkt dieses Kapitels steht die Erarbeitung von Lösungsansätzen, die zur Umsetzung der Web-Applikation herangezogen werden sollen. Es wird darum gehen, die Zielgruppe auszumachen und zu ermitteln, welche Anforderungen an die Web-Applikation gestellt werden, damit ein konzeptueller Zusammenhang und in einem weiteren Arbeitsprozess ein Datenmodell entsteht. Darüber hinaus werden Ideen zur Gestaltung der Benutzeroberfläche und zu den Abläufen der Web-Applikation präsentiert.

3.1 Zielgruppe

Die Web-Applikation soll in interkulturellen Trainings als Werkzeug zur Erfassung und Visualisierung von Multikollektivität zum Einsatz kommen. Die Zielgruppe lässt sich entsprechend aus den Teilnehmerinnen und Teilnehmern solcher Trainings definieren. Dies können studentische Gruppen bis hin zu Fach- und Führungskräfte aus international agierenden Unternehmen sein. Für die Konzeption ist es daher sinnvoll, die heterogene Zusammensetzung der Zielgruppe in die Überlegungen miteinzubeziehen. Möglich sind in diesem Zusammenhang Erwägungen zur sprachlichen Unterstützung der Applikation, zur Multi-Device-Fähigkeit und auch zur Bedienbarkeit mit unterschiedlichen, technischen Vorkenntnissen.

Wie in Abschnitt 2.1.2 erläutert, trägt das Konzept der Multikollektivität in interkulturellen Trainings bedeutend dazu bei, neue Perspektiven der Individuen auf sich selbst und die Anderen zu etablieren und so die verbindenden, gemeinsamen Elemente zu entdecken. Zu diesem Zweck sollen die Teilnehmerinnen und Teilnehmer des interkulturellen Trainings die Web-Applikation dazu nutzen, Angaben zu machen zu Interessen, Vorlieben, Dingen und Tätigkeiten, denen sie sich zugehörig fühlen. Die im Webbrowser eingegebenen Begriffe werden an die

Datenbank übertragen und für alle Teilnehmenden erfasst mit dem Ziel, Übereinstimmungen und Gemeinsamkeiten innerhalb der Gruppe zu finden. Diese sollen so visualisiert werden, dass sich durch die graphische Darstellung und die Möglichkeit der interaktiven Exploration der Ergebnisse kollektive Zugehörigkeiten und Korrelationen erkennen lassen.

Die Konzeption der Web-Applikation beginnt im nächsten Abschnitt mit der Formulierung der grundlegenden Anforderungen und Funktionen. Aus Gründen der besseren Lesbarkeit wird nachfolgend auf die gleichzeitige Verwendung der weiblichen und männlichen Schreibweise der Bezeichnung Benutzer bzw. Moderator verzichtet. Angesprochen sind sowohl weibliche als auch männliche Personen.

3.2 Grundlegende Anforderungen

Bereits in der Einleitung und den Ausführungen zur Motivation und den Zielen der vorliegenden Bachelorarbeit ist der Bezug zu dem geplanten Forschungsprojekt von Herrn Prof. Dr.-Ing. Hartmut Schirmacher und Frau Prof. Dr. Stefanie Rathje hergestellt worden. Da beide die betreuenden Lehrkräfte dieser Bachelorarbeit sind und Frau Prof. Dr. Stefanie Rathje selbst auch interkulturelle Trainings leitet, ging es zunächst darum, durch gemeinsame Gespräche eine Vorstellung von der zu entwickelnden Anwendung zu erhalten. Nach diesen Gesprächen besteht nun eine Arbeitsgrundlage, auf der die Anforderungen an die Web-Applikation ermittelt und beschrieben werden können.

Ein wesentliches und umfassendes Merkmal der Web-Applikation wird sein, dass sie einen ersten Prototypen zur Überprüfung der entwickelten Ideen darstellen soll, zum Experimentieren, ob sich die gewählten Werkzeuge für die Aufgabe der Anwendung eignen. Es soll eine Grundlage geschaffen werden, verschiedene Visualisierungs- und Interaktionsmöglichkeiten auszuprobieren, um die so gewonnenen Erkenntnisse zu nutzen und darauf aufbauend, zu einer tragfähigen und einsetzbaren Web-Applikation zu kommen.

Die folgende Definition von Kriterien, welche die Web-Applikation erfüllen soll, dient als Basis für den weiteren konzeptionellen Entwurf. Hierbei erfolgt keine genaue Priorisierung, vielmehr wird zwischen unverzichtbaren und optionalen Kriterien unterschieden. Dies erlaubt bei der Suche nach möglichen Lösungen zur Realisierung des Prototypen zunächst diejenigen Eigenschaften in Betracht zu ziehen, die maßgeblich die Funktionsweise bestimmen, ohne dabei jedoch Wunschkriterien aus der Konzeption auszuklammern.

Die nachstehenden Punkte sind als wesentliche Kriterien zu betrachten:

- Der Benutzer registriert sich bei der erstmaligen Benutzung über einen Anmelde-Dialog mit Name (realer Name oder Benutzername), Emailadresse und Kennwort. Diese Angaben sind zwingend erforderlich. Bereits registrierte Benutzer können sich mit ihren Zugangsdaten anmelden.
- Es soll neben der Benutzerrolle auch eine Rolle für einen Moderator geben.
- Die Daten der einzelnen interkulturellen Trainings werden separat erfasst. Die korrekte Zuordnung und Trennung der Daten wird durch eine eindeutige Referenzierung (z.B. ID, Titel, Gruppenname o.ä. für jedes Training) gewährleistet. Diese Angaben sind zwingend erforderlich.
- Der Moderator kann Daten zu Trainingseinheiten hinzufügen oder löschen
- Vorgegebene Kategorien dienen dem Benutzer bei der Eingabe zur Orientierung und als Impulsgeber. Der Benutzer kann die Begriffe für die verschiedenen Kategorien jedoch in wahlfreier Reihenfolge eingeben und Kategorien können auch ausgelassen werden.
- Der Moderator kann Kategorien hinzufügen oder löschen
- Die Begriffe können aus mehreren Wörtern bestehen und werden in der Datenbank erfasst. Groß- und Kleinschreibung wird bei der Prüfung auf Gleichheit ignoriert.
- Die Begriffe können durch den Benutzer (hierarchisch) verfeinert werden.
- Der Benutzer kann von ihm selbst eingegebene Begriffe löschen.
- Die durch den Benutzer eingegebenen Informationen werden durch ein ansprechendes und aussagekräftiges Layout visualisiert.
- Die Visualisierung reagiert dynamisch auf neue Daten.
- Der Benutzer kann zwischen verschiedenen Ansichten wählen, um eine spezifische Darstellung zu erhalten:
  - Eine Einzelsicht visualisiert ausschließlich den Benutzer selbst und die von ihm eingegebenen Begriffe.
  - Eine 1:1 Visualisierung basiert auf den Begriffs-Übereinstimmungen zwischen dem Benutzer selbst und einem anderen aus der Gruppe gewählten Benutzer.
  - Eine Gesamtvisualisierung stellt alle Benutzer und ihre Gemeinsamkeiten dar.

- Die Web-Applikation unterstützt die englische Sprache.
- Ein intuitives Design ermöglicht die einfache Erfassung und Handhabung der Bedienelemente.

Als optionale Wunschkriterien gelten die folgenden Anforderungen:

- Nutzungsfreundliche Erläuterungen und Bedienungshilfen unterstützen (unerfahrene) Benutzer bei der Anwendung.
- Die Benutzeroberfläche soll so gestaltet sein, dass die Nutzung der Web-Applikation auch für mobile Geräte geeignet ist.
- Zur Unterstützung der visuellen Exploration ist eine Anzeige der Visualisierung im Vollbildmodus möglich.
- Der Benutzer hat die Möglichkeit, einen Beispieldatensatz zur Anschauung zu laden.
- Die Visualisierung kann als PDF exportiert werden.

3.3 Datenmodellierung

Ausgehend von den grundlegenden Anforderungen und mit Blick auf das Ziel des zu entwickelnden Prototyps der Web-Applikation widmet sich dieser Abschnitt dem Entwurf eines geeigneten Datenmodells. Bei der Modellierung geht es darum, einen Ausschnitt aus der realen Welt zu abstrahieren, um diesen vereinfacht zu beschreiben und in eine nutzbare Struktur zu bringen [vgl. RWE13, S. 25]. Für den Einsatzbereich der Web-Applikation in einem interkulturellen Training stellt sich daher die Frage, welche Daten erfasst werden müssen und wie man diese sinnvoll strukturiert in der Datenbank ablegen kann, damit die erforderliche Bearbeitung und Abfrage der Daten möglich ist.

Ähnlich einem Entwurf auf einem Whiteboard, dient eine einfache Skizze (siehe Abbildung 3.1) dazu, die relevanten Gegenstände und Zusammenhänge zu identifizieren, gleichsam das „Problem“ zu beschreiben, um daraus Hinweise für das Datenmodell abzuleiten. Von Interesse sind demnach Elemente wie „Interkulturelles Training“, „Teilnehmer“, „Trainer“, „Begriff“ und „Kategorie“ sowie die jeweiligen Relationen.

Bei der Datenmodellierung für eine Graphdatenbank wie Neo4j (siehe Abschnitt 2.3.3) kann man sich eine Eigenschaft zunutze machen, die als "whiteboard friendly"[RWE13, S. 26]

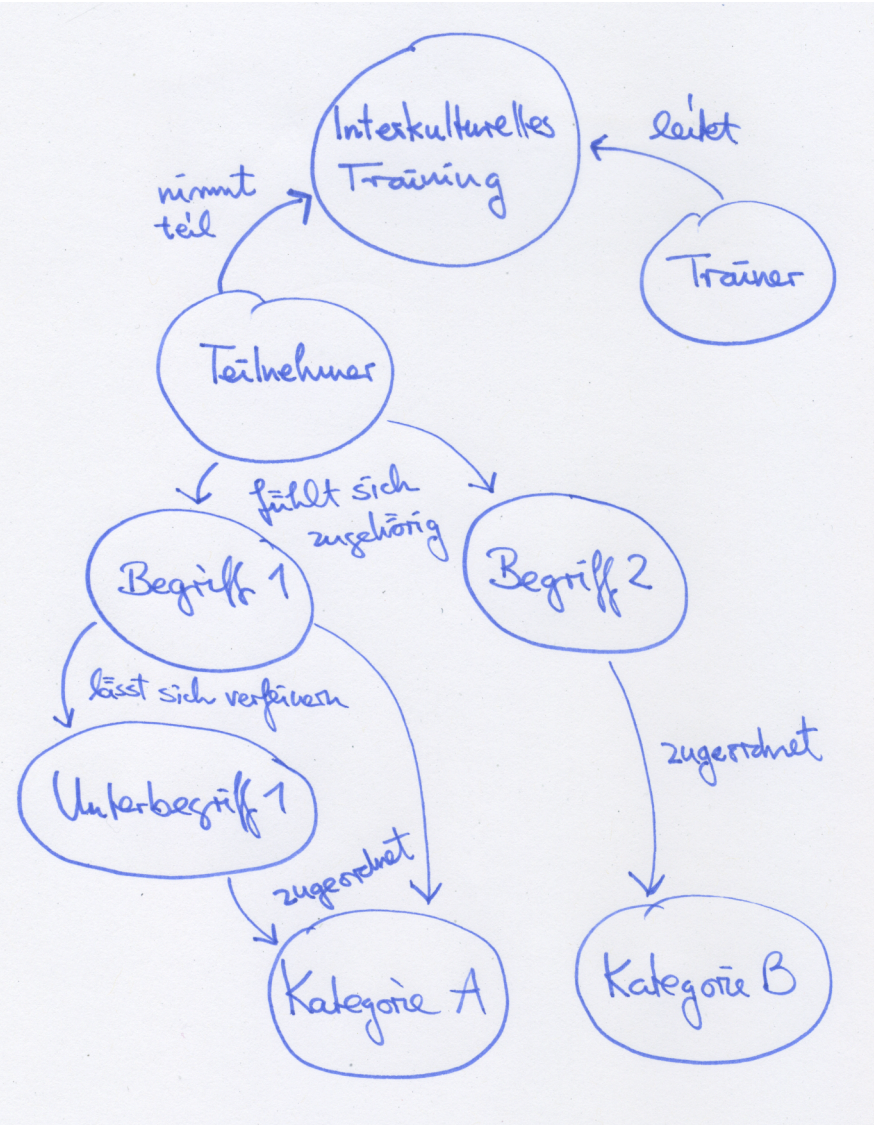


Abbildung 3.1: Datenmodell-Skizze

bezeichnet wird. Bereits die vereinfachte Darstellung des Sachverhalts in Abbildung 3.1 stellt einen Graph dar und somit lassen sich die aus der Realwelt abstrahierten *Entitäten* mit ihren Relationen direkt in das Property-Graph-Modell (vgl. Abschnitt 2.3.3.1) von Neo4j übertragen. Abbildung 3.2 zeigt einen exemplarischen Property-Graph mit den benötigten Knoten, Kanten (Relationen) und Eigenschaften und veranschaulicht das Datenmodell, wie es auch innerhalb Neo4j implementiert werden soll.

Die Knoten repräsentieren die Entitäten, die im Kontext der Anwendung wichtig sind, z.B. der Benutzer und der von ihm eingegebene Begriff. Über die Eigenschaften der Knoten werden die erforderlichen Informationen erfasst. So kann später beispielsweise in der Datenbank nach

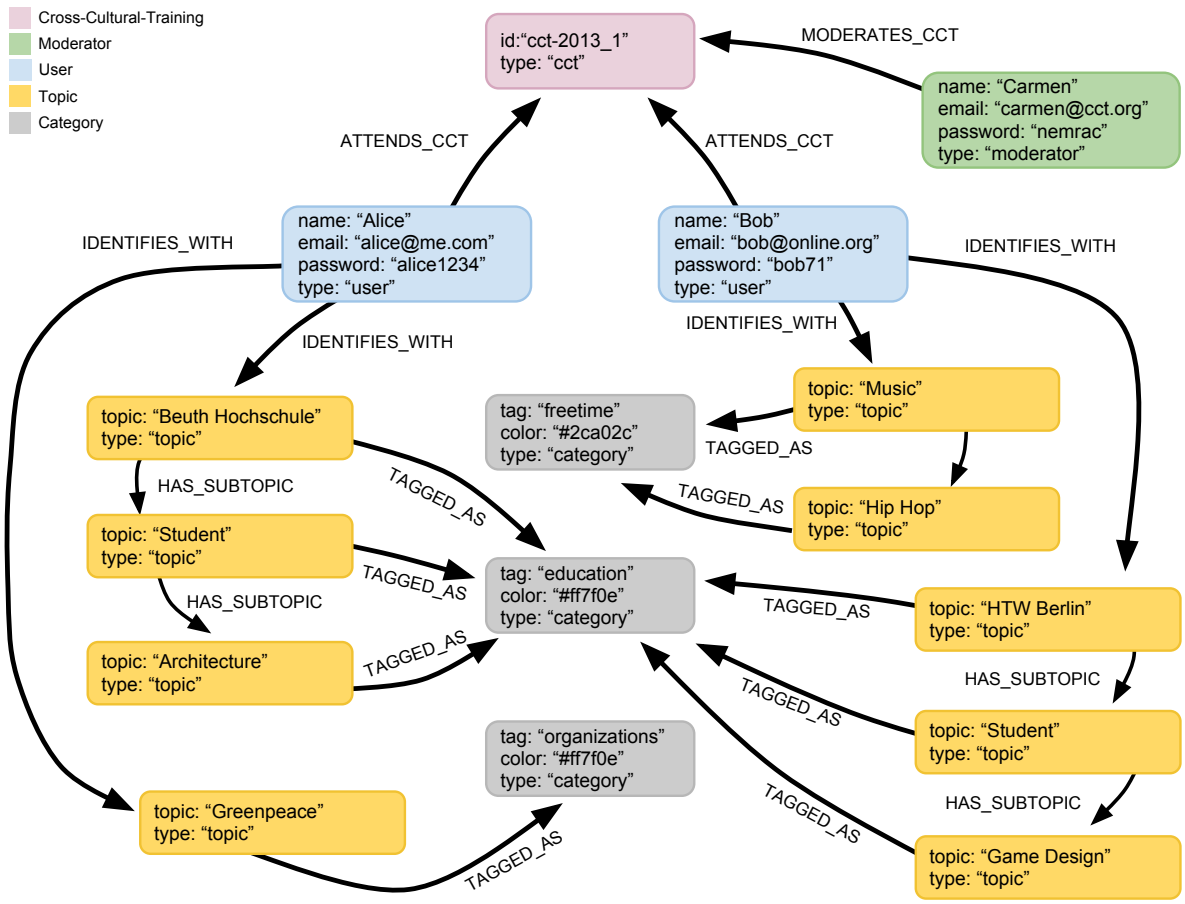


Abbildung 3.2: Exemplarischer Property-Graph

einem Benutzer mit dem Namen "Bob"gesucht werden. Mit den Kanten und dem jeweiligen Typ/Label werden die Relationen ausgedrückt, es wird Struktur und Bedeutungszusammenhang hergestellt sowie die Grundlage geschaffen für die Graphdatenoperationen mittels Cypher [vgl. RWE13, S. 65].

Jedes interkulturelle Training repräsentiert einen Property-Graph, daher lassen sich die zu erzeugenden Knoten mit ihren Eigenschaften und Relationen wie folgt zusammenfassen:

- Training
  - Eigenschaften: `id`, `type`
  - eingehende Relationen: `MODERATES_CCT`, `ATTENDS_CCT`
- Benutzer
  - Eigenschaften: `name`, `email`, `password`, `type`
  - ausgehende Relationen: `ATTENDS_CCT`, `IDENTIFIES_WITH`,

- Moderator
  - Eigenschaften: `name`, `email`, `password`, `type`
  - ausgehende Relation: `MODERATES_CCT`
- Begriff
  - Eigenschaften: `topic`, `type`
  - eingehende Relationen: `IDENTIFIES_WITH`, `HAS_SUBTOPIC`
  - ausgehende Relationen: `HAS_SUBTOPIC`
- Kategorie
  - Eigenschaften: `tag`, `color`, `type`
  - eingehende Relationen: `TAGGED_AS`

Mit dem entwickelten Datenmodell sind nun weitere Voraussetzungen geschaffen, die für den Entwurf des Prototyps wichtig sind und in die folgenden Überlegungen und Ideen zur Konzeption der Benutzeroberfläche und zu den Abläufen miteinbezogen werden.

3.4 Benutzeroberfläche und Abläufe

Die Gestaltung der Benutzeroberfläche richtet sich nach dem für Web-Applikationen typischen Aussehen und deren Einteilung in zwei Hauptbereiche: in einen schmalen Seitenbereich, zumeist für Eingaben oder Navigation vorgesehen und in einen großen Hauptbereich, der die restliche zur Verfügung stehende Seitenbreite für die Anzeige des Inhalts einnimmt. Abbildung 3.3 zeigt Google Maps stellvertretend als eine der bekanntesten Web-Applikationen.

Nachfolgend soll mithilfe von *Wireframes* der konzeptionelle Entwurf der Benutzeroberfläche erfolgen. Ein Wireframe kann als schematische Skizze oder Bauplan einer Webseite verstanden werden und beschreibt das Seitenlayout oder die Anordnung des Inhaltes. Darin enthalten sind üblicherweise die Elemente der Benutzeroberfläche und die Navigationslogik, um damit zu veranschaulichen, wie diese zusammenarbeiten. Wireframes helfen dabei, die Funktionalität und den Ablauf sowie die Beziehungen zwischen den verschiedenen Bereichen einer Webseite zu bestimmen [vgl. Wir]. Da bereits festgelegt ist, das Front-End/CSS-Framework Bootstrap zu verwenden, fließen entsprechende Inspirationen aus den Bootstrap-Beispielen in die Konzeption ein. Weitere Anregungen liefert das Buch „Single Page Web Applications“ [siehe MP13], denn die Web-Applikation soll als *Single Page Application* entwickelt werden, d.h. sie besteht aus

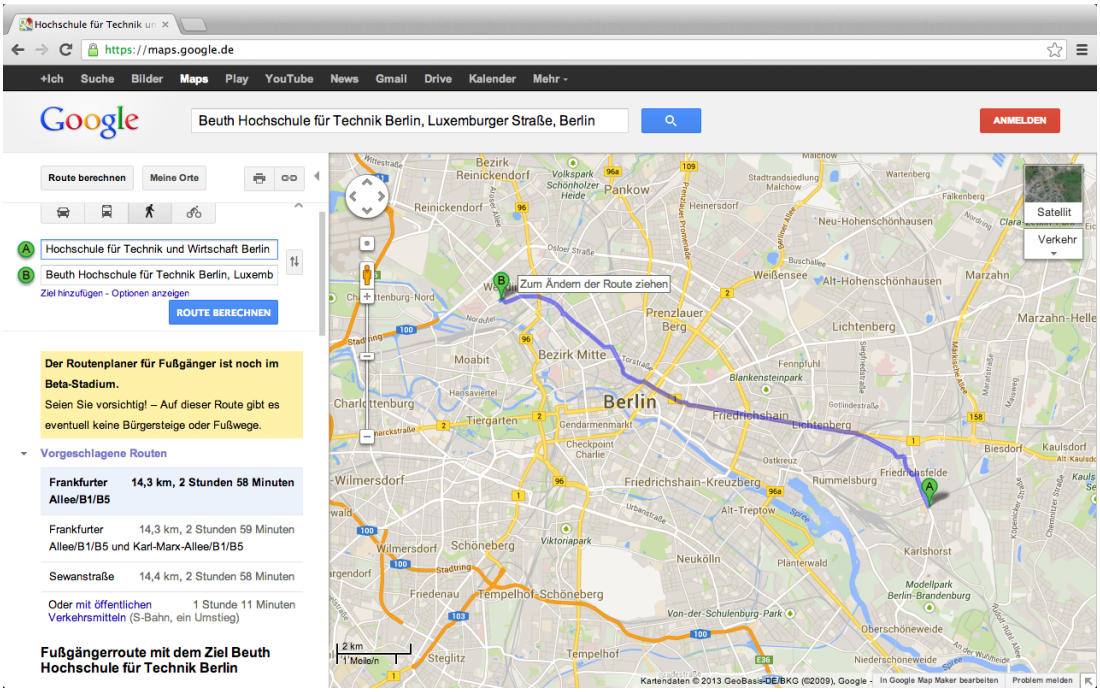


Abbildung 3.3: Google Maps als Beispiel für das typische Layout einer Web-Applikation (Bildschirmfoto)

einer einzigen HTML-Seite, deren Benutzeroberfläche wie oben geschildert in spezifische Bereiche gegliedert wird.

Die Beschreibungen der Benutzeroberfläche erfolgen aus der Sicht des Benutzers, gleichzeitig lassen sich aus dem Entwurf aber auch die funktionalen Abläufe sowie die umzusetzenden Prozesse in Bezug auf die Datenbank benennen. Variationen für mobile Geräte finden in den Wireframes keine Berücksichtigung. Der Entwurf des Prototyps soll vorerst auf eine Desktop-PC Applikation ausgerichtet sein und es ist zu testen, inwieweit mit dem Einsatz von Bootstrap die parallele Umsetzung auch für kleinere Bildschirmgrößen möglich ist.

Bevor die wesentlichen Seitenbereiche im Einzelnen erläutert werden, soll Abbildung 3.4 zunächst die Idee der geplanten Gliederung der Benutzeroberfläche vermitteln.

3.4.1 Registrierung und Anmeldung

Beim erstmaligen Ansteuern der Web-Applikation soll ein Bereich für die Registrierung und Anmeldung des Benutzers angezeigt werden, die Idee dafür ist dem Wireframe in Abbildung 3.5 zu entnehmen.

Sofern sich ein Benutzer noch nicht registriert hat, wird dieser nach der Eingabe der notwendigen Angaben (Name, Emailadresse und Passwort) und dem Klick auf den „Sign Up”-Button

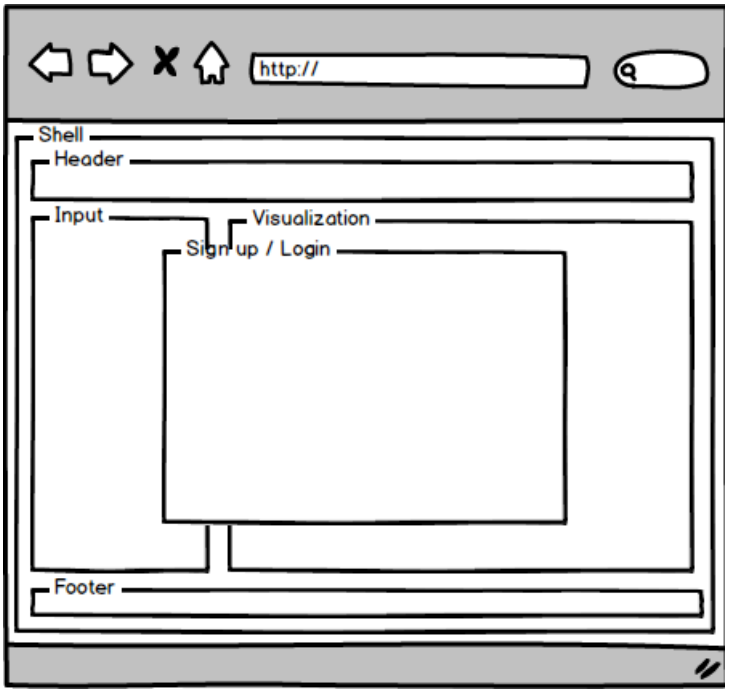


Abbildung 3.4: Wireframe für die Gliederung der Benutzeroberfläche [vgl. Abbildung aus MP13, S. 98]

als neuer Benutzer registriert (in Abbildung 3.5 links zu sehen). In der Datenbank wird somit ein neuer Benutzer-Knoten angelegt. Alternativ können sich bereits registrierte Benutzer direkt mit ihren Anmeldedaten einloggen. Hierbei hat eine Abfrage der Datenbank anhand der eingegebenen Daten zu erfolgen. Typische Platzhaltertexte unterstützen den Benutzer bei der Eingabe der entsprechenden Daten und darüber hinaus bietet es sich an, auch eine „Passwort vergessen“-Funktion vorzusehen. Nach einer erfolgreichen Registrierung bzw. Anmeldung wird dieser Anmelde-Dialog ausgeblendet und der Benutzer hat fortan Zugriff auf die Anwendung. Im Header-Bereich ändert sich gleichzeitig auch die Anzeige des Login-Status. Dies gilt analog, wenn sich der Benutzer abmeldet.

3.4.2 Benutzereingabe

Der Bereich für die Benutzereingabe soll zunächst am Beispiel für den Benutzer erläutert werden, da sich hier Unterschiede zwischen Benutzer und Moderator ergeben. Abbildung 3.6 zeigt die mögliche Gestaltung dieses Bereichs anhand eines ausklappbaren Menüs mit Einträgen zum Training und zu den vorgegebenen Kategorien.

Der Benutzer kann hier das jeweilige Menüelement zum Anzeigen des Inhalts anklicken und dann in dem aufgeklappten Bereich mit der Eingabe beginnen. Für eine korrekte und

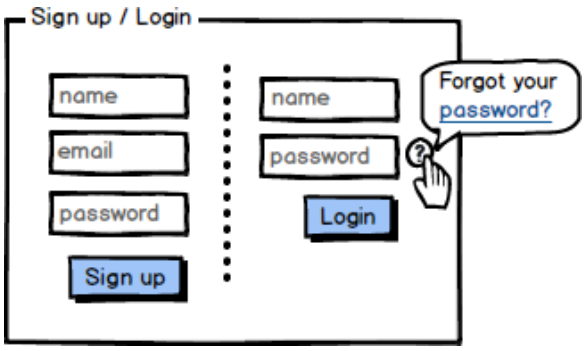


Abbildung 3.5: Wireframe für den Anmelde-Dialog

eindeutige Zuordnung des Benutzers zu einem Training ist diese Angabe zu machen *bevor* die Begriffe eingegeben werden. Idealerweise kann der Benutzer daher aus dem ausgeklappten Menüeintrag „Training“ die entsprechende Information, die dynamisch aus der Datenbank gewonnen wird, einer Auswahlliste entnehmen und selektieren. Gemäß dem Datenmodell soll hierdurch in der Datenbank die Relation zwischen dem Benutzer-Knoten und dem Trainings-Knoten hergestellt werden. Dieser Vorgang ist links in Abbildung 3.6 angedeutet, wohingegen rechts der aufgeklappte Zustand eines Menüeintrags „Category B“ zu sehen ist. Auch hier sollen später die Kategoriebezeichnungen dynamisch als Menüeintrag generiert werden.

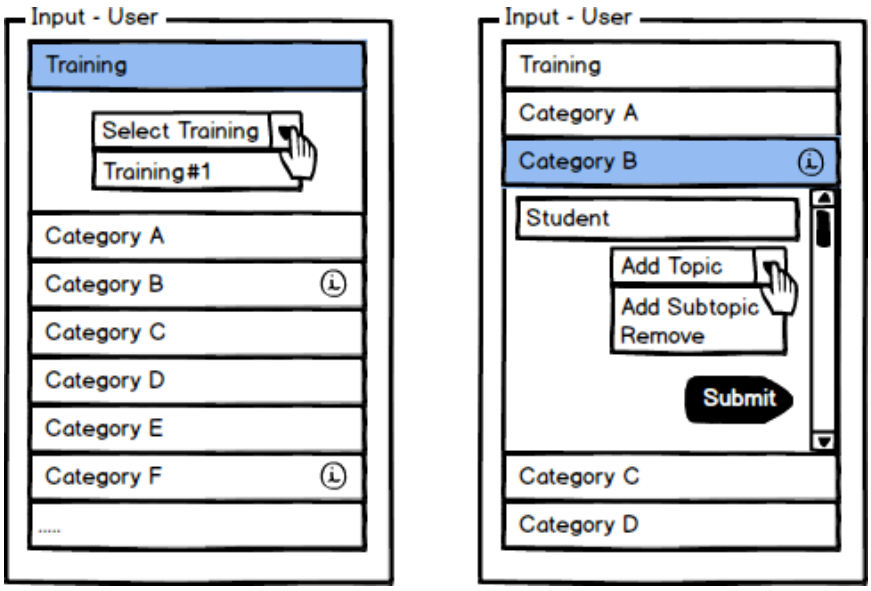


Abbildung 3.6: Wireframe für die Eingabe des Benutzers

Mit der Eingabe mehrerer Begriffe und gegebenenfalls Unterbegriffe zu einer Kategorie wächst

die Anzahl der Eingabefelder, wodurch der gesamte Bereich verlängert und unter Umständen unübersichtlich wird. Dies kann durch das Ein- und Ausklappen der Inhalte umgangen werden. Damit der Benutzer je nach Bedarf ein weiteres Eingabefeld zur Verfügung hat, muss dieses auf Anforderung dynamisch hinzugefügt werden. Der Lösungsansatz in Abbildung 3.6 zeigt eine Auswahlliste für das Hinzufügen eines Begriffs („Add Topic“) und das Hinzufügen eines Unterbegriffs („Add Subtopic“). Mit der Selektion eines dieser Einträge wird ein neues, leeres Eingabefeld erzeugt und dabei auch die Hierarchie berücksichtigt. Außerdem kann der Benutzer ein Eingabefeld auch wieder entfernen („Remove“). Durch das Klicken auf den „Submit“-Button werden in der Datenbank für jedes vom Benutzer ausgefüllte Eingabefeld ein Begriff-Knoten und gleichzeitig alle seine Relationen erzeugt. Als optionale Ergänzung ist das kleine „Info“-Symbol zu sehen, das dem Benutzer erläuternde Informationen zu einer Kategorie anzeigen könnte.

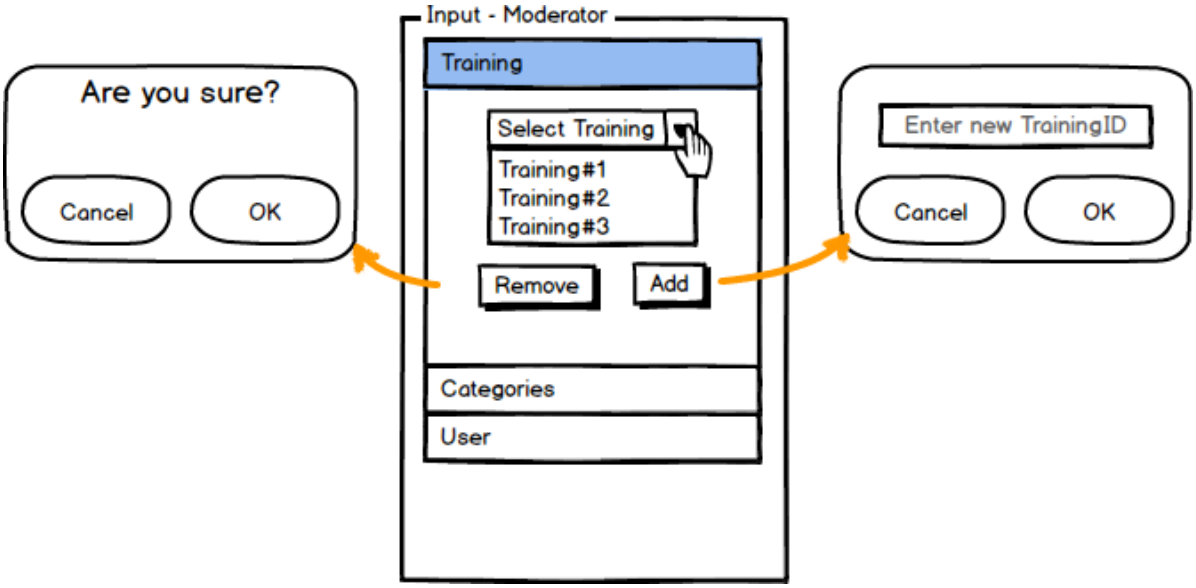


Abbildung 3.7: Wireframe für die Eingabe des Moderators

Für die Aufgaben des Moderators ist eine andere Anordnung des Eingabebereichs erforderlich. Der Moderator soll hier administrative Möglichkeiten in Bezug auf das Training, die Kategorien und Benutzer haben und so kann der Moderator jeweils den zu bearbeitenden Eintrag aus dem ausklappbaren Menü wählen. Abbildung 3.7 zeigt exemplarisch den aufgeklappten Zustand des Menüeintrags „Training“, der dem Moderator die Bedienelemente anzeigt, um ein Training zu löschen oder hinzuzufügen. Bei einem Löschvorgang soll durch eine zusätzliche Bestätigung über einen Dialog sichergestellt werden, dass es zu keiner versehentlichen Löschung der Daten

kommt. Diese Abläufe sind identisch für die Menüeinträge „Training“, „Categories“ und „User“ und erfordern die entsprechenden Datenbankoperationen.

3.4.3 Visualisierung und interaktive Exploration

Damit der Benutzer zur Anzeige der Visualisierung zwischen den drei geplanten Betrachtungsmodi Einzelsicht, 1:1-Ansicht und Gesamtvisualisierung wählen kann, soll dieser Bereich der Benutzeroberfläche unter Verwendung von *tabs* (dt. *Registerkarten*) gestaltet werden. Sobald ein Tab aktiv ist, soll die entsprechende Visualisierung eingeblendet werden. Für die Einzelsicht und die Gesamtvisualisierung gilt dabei, dass jeweils direkt bei aktivem Tab die Datenbankabfrage erfolgt oder bereits erfolgt ist. Das Beispiel in Abbildung 3.8 soll daher die Betrachtung einer möglichen 1:1-Visualisierung veranschaulichen, da hier noch ein Zwischenschritt erforderlich ist.

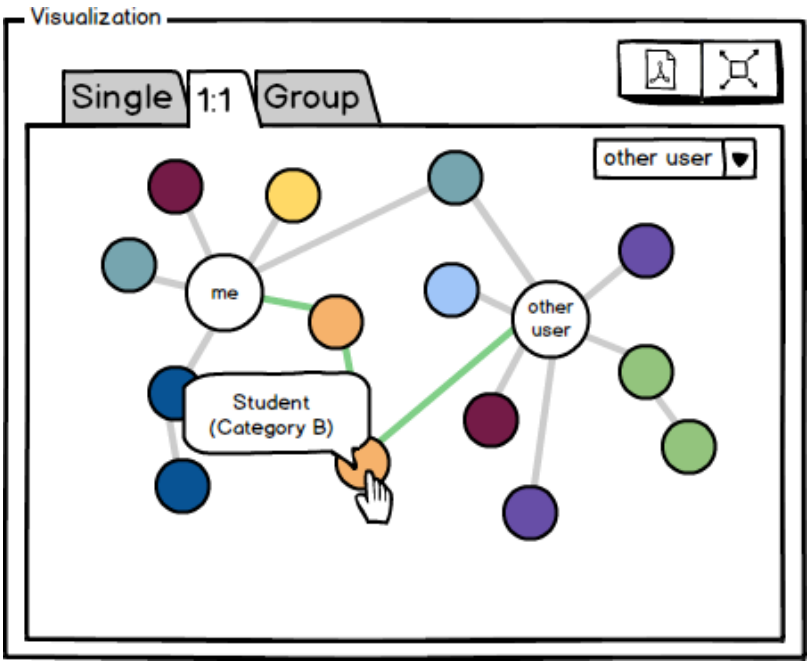


Abbildung 3.8: Wireframe für die Anzeige der Visualisierung

Im rechten, oberen Bereich befindet sich ein Auswahllisten-Element, aus dem der Benutzer eine andere Person aus der Gruppe wählen kann. Die Liste der Benutzer soll dynamisch aus der Datenbank gewonnen werden und alle Benutzer-Knoten beinhalten, die eine Relation zu dem gleichen Trainings-Knoten aufweisen. Damit die Liste den Benutzer selbst nicht ebenfalls aufführt, soll dieser dabei ausgeklammert werden. Sobald der Benutzer eine andere Person

aus der Liste ausgewählt hat, muss die Auswertung der Begriffe erfolgen und entsprechend visualisiert werden. Für die Umsetzung der Visualisierung ist es hier von Interesse, dass bereits durch farbliche Codierung, die einen Rückschluss auf die Kategorien zulässt (jeder Kategorie-Knoten hat eine Eigenschaft `color`), eine visuelle Exploration angestossen werden kann. Der Benutzer soll weiterhin die Möglichkeit haben, die Visualisierung interaktiv zu erkunden, um wie in Abbildung 3.8 angedeutet durch Mausinteraktionen zusätzliche Informationen und eine Hervorhebung der Gemeinsamkeit zu erhalten oder beispielsweise zu zoomen oder die Visualisierung besser zu arrangieren.

Ein separates Wireframe-Beispiel für den Moderator wird an dieser Stelle vernachlässigt. Die Unterschiede bestehen in einer fehlenden Einzelsicht und einer zusätzlichen Auswahlmöglichkeit bei einer 1:1-Visualisierung, damit der Moderator zwei Benutzer aus der Gruppe auswählen und betrachten kann.

Kapitel 4

Technische Umsetzung

In diesem Kapitel soll die Implementierung der Web-Applikation, die anhand der zuvor entwickelten Lösungsideen erfolgt, beschrieben werden. Dabei spielt zunächst der grundlegende Aufbau des Gesamtsystems eine Rolle und welche Aufgabe die Systemkomponenten haben. Im weiteren Verlauf werden anhand von Beispielen die angewandten JavaScript Entwurfsmuster erläutert wie auch die Frameworks, die zur Umsetzung des Prototyps beitragen.

4.1 Client-Server-Architektur

Damit die Web-Applikation funktionsfähig ist, werden drei Komponenten benötigt. Diese bilden das Gesamtsystem und bestehen aus dem Webserver, dem Neo4j Datenbankserver und dem Client (Webbrowser) und werden nachfolgend ausführlich vorgestellt. Abbildung 4.1 zeigt die Systemarchitektur und veranschaulicht die Interaktion, die zwischen den Komponenten der Anwendung abläuft.

4.1.1 Webserver

Mit der Entscheidung eine rein clientseitige Web-Applikation zu entwickeln und für diesen Prototyp spezifische Aufgaben eines Servers wie Autorisierung und Authentifizierung zu vernachlässigen, ergibt sich für die Komponente des Webservers die Konsequenz, dass dieser lediglich dazu dient, dem Client alle benötigten JavaScript-, CSS- und HTML-Dateien zur Verfügung zu stellen. In der Entwicklungsphase übernimmt diese Aufgabe ein lokal eingerichteter *Apache-Webserver*[20] unter der Adresse `http://localhost:8888/`.

[20] Der Apache-Webserver ist Bestandteil des Softwarepakets *MAMP* (ein Akronym für *Macintosh*, *Apache*, *Mysql* und *PHP*), das die Einrichtung einer lokalen Serverumgebung auf einem Mac OS X Rechner ermöglicht. Vgl. *http://www.mamp.info/de/mamp/index.html* (besucht am 20.08.2013).

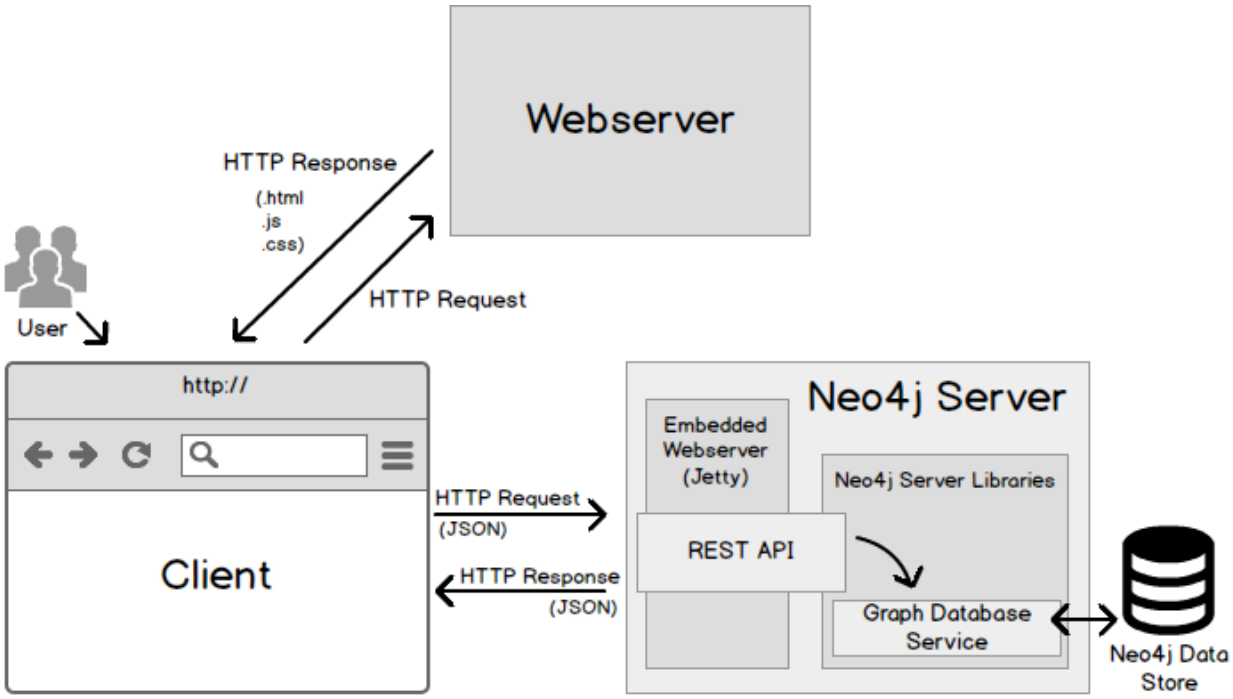


Abbildung 4.1: Systemarchitektur [vgl. Abbildung aus PVW13, S. 167].

4.1.2 Neo4j Server

Als Neo4j Server kommt während der Umsetzung der Web-Applikation die *Neo4j 1.9.1 Community Edition* zum Einsatz, die für nicht-kommerzielle Projekte frei verfügbar ist. Nach der Installation ist der Neo4j Server unter der Adresse `http://localhost:7474/`[21] zu erreichen. Unter `http://localhost:7474/db/data` befindet sich der sogenannte *service root* der REST API – dies ist der Einstiegspunkt, um die Schnittstelle zu nutzen.

REST steht für *REpresentational State Transfer*[22] und für einen Software-Architekturstil, der unter anderem beschreibt, dass man sich die Art und Weise, wie das World Wide Web funktioniert, auch bei der Bereitstellung von Webanwendungen oder Webservices zunutze machen kann. Demnach ist es möglich, beliebige Ressourcen (nicht nur herkömmliche Webseiten) zugänglich zu machen und in einem lesbaren Datenformat bereitzustellen, indem man diese mit einem HTTP `GET`-Request über ihren *URL (Uniform Ressource Locator)* anfordert. Erweitert um die übrigen HTTP-Methoden `POST`, `PUT`, `DELETE` können Ressourcen auch hinzugefügt,

[21] Alternativ kann auch die IP-Adresse des localhost verwendet werden: `http://127.0.0.1:7474/` (siehe Abbildung 4.2)

[22] Die Theorie zu REST hat Roy Thomas Fielding im Rahmen seiner Doktorarbeit „*Architectural Styles and the Design of Network-based Software Architectures*“ entwickelt. Das entsprechende Kapitel ist zu finden unter *http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm* (besucht am 20.08.2013).

geändert oder gelöscht werden. Im Falle von Neo4j bestehen die Ressourcen in Knoten und Relationen und das Datenaustauschformat ist JSON [vgl. PVW13, S. 168-169]. Ein anderes Beispiel für einen sogenannten *RESTful* Service ist die *Twitter REST API*[23] mit Ressourcen wie *tweets* oder *followers*.

Um die Knoten der Graphdatenbank anzusprechen, benutzt man somit beispielsweise den URL `http://localhost:7474/db/data/node`. Relevant für die Umsetzung ist jedoch die Cypher-Adresse `http://localhost:7474/db/data/cypher`. Damit ist es möglich, auf REST basierende Graphdatenbankoperationen mit Cypher durchzuführen. Die Cypher-Anweisung wird dabei als JSON übergeben, deshalb ist es wichtig, dies im HTTP-Request anzugeben, ebenso für die Daten, die von Neo4j als JSON zurückgegeben werden. Implementiert wird dies mit der `ajax()`-Funktion, die jQuery bereitstellt (siehe Listing 4.2). Für die Übergabe der Daten lautet die Angabe `Content-Type:application/json` und für die erwartete Datenrückgabe `Accept:application/json`.

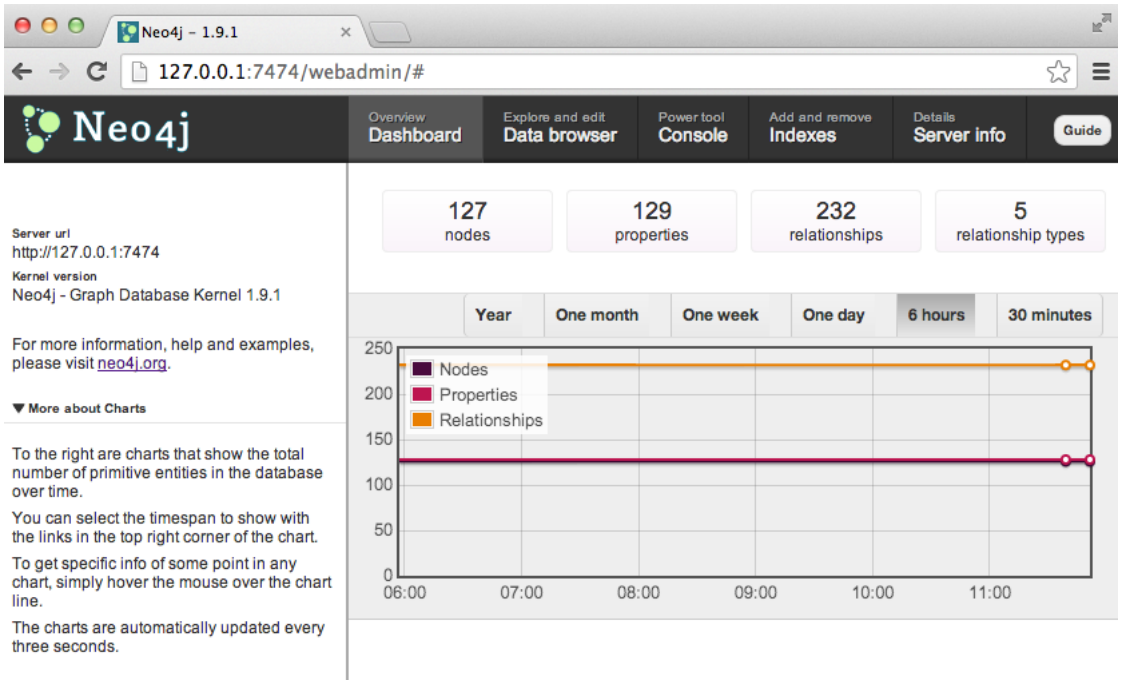


Abbildung 4.2: Integrierter Administrationsbereich des Neo4j Servers (Bildschirmfoto – Ausschnitt).

Unter `http://localhost:7474/webadmin/` kann der integrierte Administrationsbereich des Neo4j Servers aufgerufen werden. Abbildung 4.2 zeigt die Einstiegsseite des Administrationsbereichs, die eine sich ständig aktualisierende Grafik bereithält, aus der die Zahl aller Knoten, Relationen und Eigenschaften im Verlauf der Zeit zu entnehmen ist. Darüber hinaus gibt es

[23] vgl. *https://dev.twitter.com/docs/api/1.1* (besucht am 21.08.2013)

weitere Bereiche, die mit Informationen und Bearbeitungsmöglichkeiten aufwarten. Interessant sind hier insbesondere die Bereiche „Data browser“ und “Console“.

Mit dem „Data browser“ (siehe Abbildung 4.3) lassen sich bequem Knoten und Relationen erzeugen und löschen sowie Eigenschaften bearbeiten. Ebenso wird dort eine Visualisierung des Graphen angeboten. Diese Möglichkeit führt im Bezug auf die Implementierung der erforderlichen Funktionalität für den Moderator zu der Entscheidung, die Priorität auf die Realisierung der Benutzer-Funktionen zu setzen und sich aus Zeitgründen zunächst darauf zu konzentrieren.

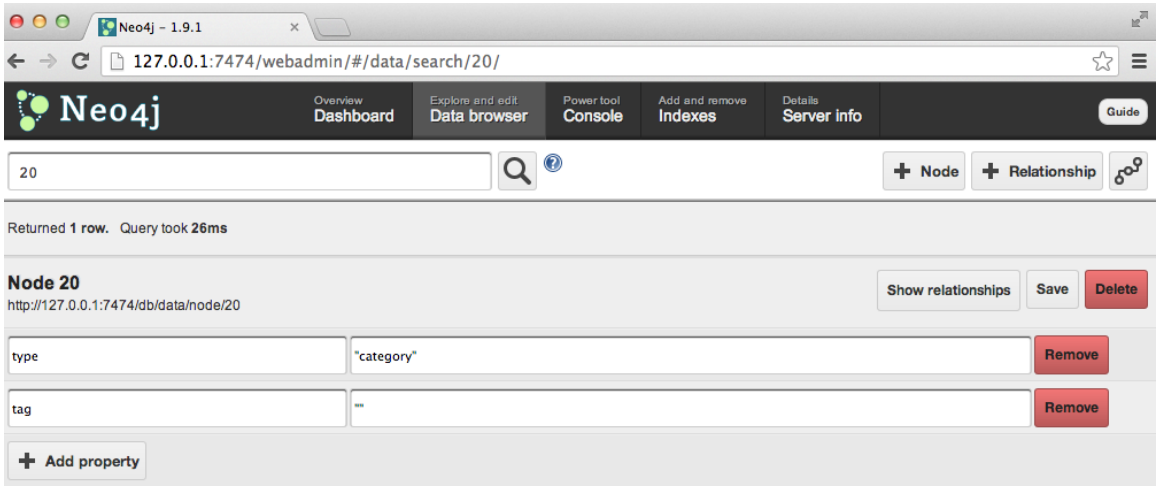


Abbildung 4.3: Neo4j Administrationsbereich – „Data browser“ (Bildschirmfoto – Ausschnitt).

Innerhalb des Bereichs „Console“ stehen nützliche Werkzeuge zur Verfügung, um die Abfragesprache Cypher zu testen oder mit gültigen Cypher-Anweisungen ebenfalls wie mit dem „Data browser“ Datenbankoperationen vornehmen zu können. Hierzu nutzt man die „Neo4j Shell“. Daneben gibt es noch die „HTTP-Konsole“ und eine weitere Konsole zur Verwendung von *Gremlin*, einer Sprache zur Traversierung von Graphen, auf die an dieser Stelle jedoch nicht näher eingegangen werden kann.[24] Die „HTTP-Konsole“ (siehe Abbildung 4.4) wird während der Entwicklung dazu herangezogen, die RESTbasierten HTTP-Requests zu testen, die aus der Web-Applikation an den Neo4j Server gesendet werden müssen und leistet hierbei eine wertvolle Hilfestellung.

[24] Für weitere Informationen und zur Dokumentation von Gremlin siehe *https://github.com/tinkerpop/gremlin/wiki* (besucht am 21.08.2013).

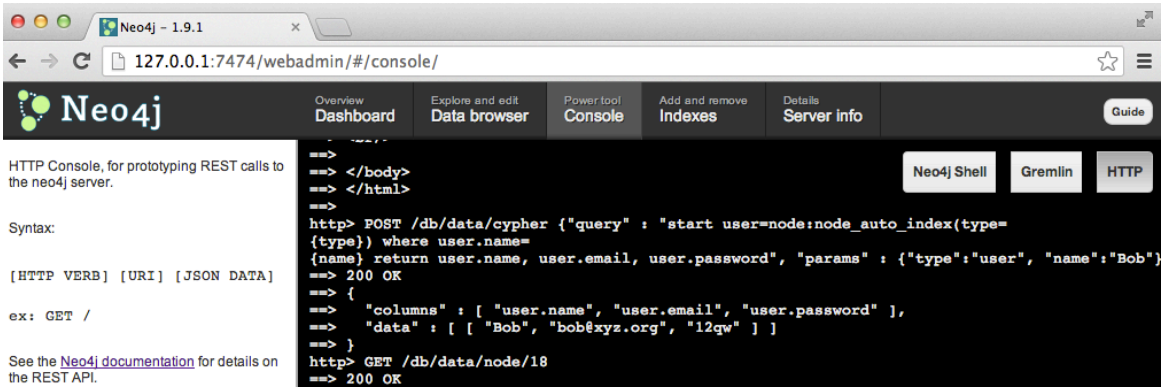


Abbildung 4.4: Neo4j Administrationsbereich – „HTTP"-Konsole (Bildschirmfoto – Ausschnitt).

4.1.3 Client

Mit dem erstmaligen Laden der Web-Applikation soll dem Client (Webbrowser) alles zur Verfügung stehen, was zum Ausführen und zur korrekten Darstellung benötigt wird. Als Single Page Application besteht die Web-Applikation aus einer einzigen statischen HTML-Seite. Der Quellcodeausschnitt in Listing 4.1 zeigt die `index.html` Datei (zur besseren Übersicht sind die Deklarationen zu den Meta-Angaben, Stylesheet- und Skripteinbindungen im Kopfbereich des Dokuments innerhalb von `<head>...</head>` ausgespart).

Listing 4.1: Die HTML–Datei index.html (vereinfacht)

```
1  <!DOCTYPE html>
2  <!-- index.html - application static html -->
3    <head>
4          <title>Cohesion Catalyst</title>
5      [...]
6          <script type="text/javascript">
7              $(document).ready(function () {
8                  app.initModule($("#app"));
9              });
10         </script>
11   </head>
12   <body>
13     <div id="app"> <!-- main container of the application -->
14     </div>
15   </body>
16 </html>
```

In dieser HTML-Datei gibt es lediglich ein `<div>`-Element mit der `id="app"` (Zeile 13), das als Hauptcontainer der Web-Applikation fungiert. Der restliche HTML-Inhalt, der die Seitenbe-

reiche strukturiert, wird dort dynamisch hinzugefügt. Sobald das Dokument geladen ist, kann dies mit der jQuery `$(document).ready()`-Funktion vollzogen werden. Die Herausforderung bei der Realisierung besteht darin, das Zusammenspiel der einzelnen Seitenbereiche, die die Benutzeroberfläche laut Entwurf (vgl. Abschnitt 3.4) enthalten soll, zu organisieren. Hierzu wird ein modularer Aufbau angestrebt, der im folgenden erklärt wird.

Da jeder Seitenbereich eine eigene Zuständigkeit repräsentiert, z.B. die Eingabe der Begriffe oder die Anzeige der Visualisierung, macht es Sinn, die jeweilige Funktionalität in separaten JavaScript-Modulen umzusetzen. Damit wird nicht nur eine logische Grenze definiert, sondern es dient auch einer besseren Übersicht während der Implementierung, wenn das Programm wächst und fördert eine gute Wartbarkeit. Hinzu kommen – nach erfolgreicher Umsetzung – erhöhte Chancen auf Wiederverwendung eines Moduls. Mikowski und Powell nennen dies „feature module strategy“ [MP13, S. 140] und vergleichen die Entwicklung von JavaScript-Modulen, die in einer Web-Applikation für den ihnen zugewiesenen Bereich zuständig sowie in sich geschlossen und möglichst unabhängig von den anderen JavaScript-Modulen sind, mit dem Einbinden eines Moduls von Drittanbietern wie beispielsweise der Dienst zum Kommentieren von Webseiten von *Disqus*[25] oder der *Google* „+1“-Button[26] [vgl. MP13, S. 141-143].

Innerhalb der JavaScript-Module wird ebenfalls auf eine Trennung der Aufgaben nach dem *Model-View-Controller* (MVC) Entwurfsmuster geachtet. Dieses Entwurfsmuster beschreibt die Gliederung einer Anwendung in drei Schichten: eine Datenschicht (Model), eine Präsentationsschicht (View) und eine Steuerungsschicht (Controller). Damit ist grundsätzlich folgender Ablauf gemeint [vgl. Mac11, S. 2-3]:

1. In der Anwendung erfolgt über die Benutzeroberfläche eine Interaktion durch den Benutzer.
2. Der Controller übernimmt die Bearbeitung dieser Interaktion.
3. Der Controller kommuniziert mit dem Model, veranlasst eine Änderung der Daten oder fordert Daten an, um das Ergebnis an die View zu liefern.
4. Die View (Benutzeroberfläche) präsentiert dem Benutzer die erhaltenen Daten.

Für jede Schicht ergibt sich somit ein Verantwortungsbereich und dieses Prinzip lässt sich auch auf der Ebene der einzelnen JavaScript-Module anwenden [vgl. MP13, S. 143-145]. Hierzu werden entsprechende Funktionen für die Handhabung und Verarbeitung der Benutzerinteraktionen sowie die Kommunikation mit der Datenschicht realisiert. Die Verwendung der Handlebars HTML-Templates unterstützt dies zusätzlich, da die HTML-Struktur der einzelnen

[25] vgl. *http://disqus.com/about/* (besucht am 21.08.2013)
[26] vgl. *https://developers.google.com/+/web/+1button/* (besucht am 21.08.2013)

Seitenbereiche in separaten `.handlebars`-Dateien erstellt wird, die bei der Initialisierung der Module dann gerendert und in die HTML-Struktur eingefügt werden (vgl. Abschnitt 2.3.2.4 sowie 4.3.2).

Tabelle 4.1 führt die für die Benutzeroberfläche relevanten JavaScript-Module mit ihren zugehörigen Handlebars-Templates auf. Die Dateinamen sollen den Zuständigkeitsbereich der Module widerspiegeln und sind parallel für die JavaScript- und Handlebars-Dateien vergeben. Daran ist zu erkennen, welche Dateien zusammengehören und die Struktur sorgt dafür, dass man sich in den Projektdateien gut zurecht finden kann. Zudem ist diese Vorgehensweise nützlich in Bezug auf eine mögliche Fortsetzung des Projekts, wenn weitere Personen hinzukommen, die den Quellcode noch nicht kennen. Die vorangestellte Kennzeichnung `app` ergibt sich aus dem Namensraum, der durch das Hauptmodul `app.js` definiert wird (siehe Abschnitt 4.2.1).

Tabelle 4.1: Browserspezifische JavaScript-Module

JavaScript-Modul	Handlebars-Template
`app.shell.js`	`app.shell.handlebars`
`app.header.js`	`app.header.handlebars`
`app.login.js`	`app.login.handlebars`
`app.topic_input.js`	`app.topic_input.handlebars`
`app.visualization.js`	`app.visualization.handlebars`

Die in der Tabelle aufgelisteten Module haben die folgenden Aufgaben:

- `app.shell.js` initialisiert die für die Interaktion mit der Anwendung erforderlichen JavaScript-Module und koordiniert diese bezüglich des Login-Status (z.B. das Ein- und Ausblenden des Login-Bereichs). Mit dem Ausdruck *shell* (dt. *Hülle, Mantel*) soll deutlich gemacht werden, dass `app.shell.js` als Umhüllung für die anderen Module dient und auch modulübergreifende Steuerungsfunktionalität besitzt [vgl. MP13, S. 87-89].
- `app.header.js` zeigt den Login-Status an und stellt eine Logout-Funktionalität bereit.
- `app.login.js` setzt die Registrierungs- und Anmeldefunktionalität um. Hier erfolgt beispielsweise auch eine Prüfung, ob der Benutzer alle Felder ausgefüllt hat (siehe Abbildung 4.5).
- `app.topic_input.js` ist für die Benutzereingabe zuständig. In diesem Modul sind unter anderem die Funktionen zum Hinzufügen und Entfernen von Eingabefeldern implementiert sowie zur Aufbereitung der eingegebenen Begriffe, bevor diese weitergegeben werden können.

- `app.visualization.js` steuert die Benutzerinteraktion zur Auswahl des gewünschten Betrachtungsmodus und ist für die Visualisierung der Daten mit D3 verantwortlich.

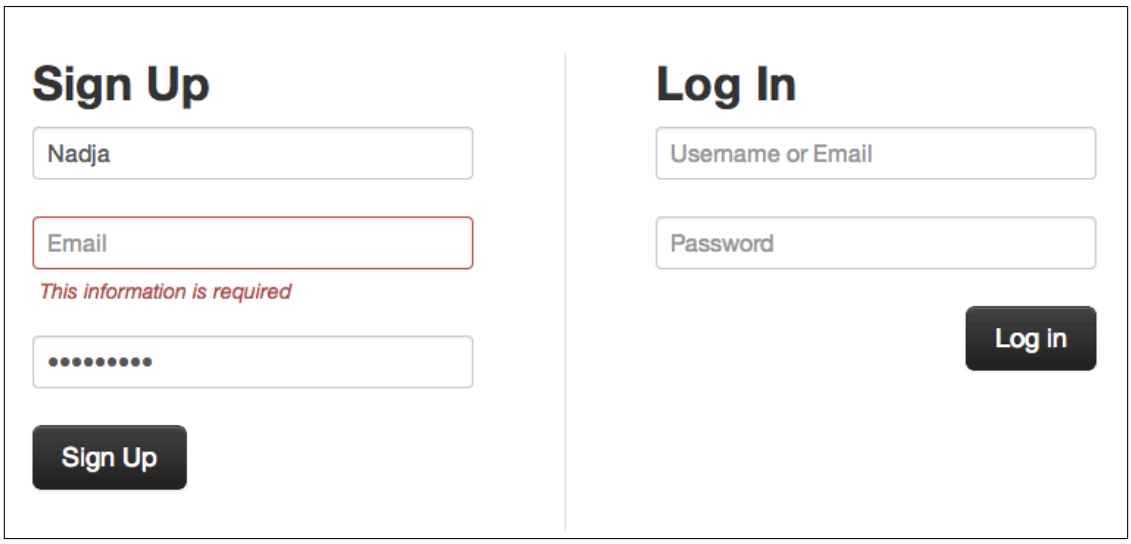


Abbildung 4.5: Umgesetzter Anmelde-Dialog mit Fehlermeldung (Bildschirmfoto – Ausschnitt).

Neben den zuvor genannten sind weitere wesentliche JavaScript-Module erforderlich, die von der Benutzeroberfläche unabhängig jedoch maßgeblich für das Datenmanagement sowie für die Kommunikation mit der Datenbank zuständig sind:

- `app.model.js` beinhaltet die Logik und die Funktionalität, um Daten zu verarbeiten oder angeforderte Daten bereitzustellen. So werden hier beispielsweise Funktionen realisiert, um ein neues Benutzerobjekt zu erzeugen (`createUser(userdata)`) oder den aktuellen Benutzer abzufragen (`getCurrentUser()`).
- `app.data.js` erhält von `app.model.js` die Daten und interagiert mit dem Datenbankserver, d.h. dieses Modul sendet die HTTP-Requests mit den Cypher-Abfragen und empfängt die HTTP-Response.

Listing 4.2 zeigt einen Ausschnitt aus dem Quellcode von `app.data.js`. Zu sehen ist die Funktion `lookUpAllUserNodes()`, die als Parameter den Namen des aktuellen Benutzers und die ID des Trainings erhält. Mit dieser Funktion soll in der Graphdatenbank nach allen Benutzerknoten gesucht werden, die eine Relation zu dem Trainingsknoten mit der übergebenen ID haben, mit Ausnahme des aktuellen Benutzers. Erforderlich ist diese Funktion, wenn ein Benutzer eine 1:1 Visualisierung betrachten möchte und dafür eine Auswahl der anderen Benutzer benötigt. Innerhalb dieser Funktion wird nun mittels Cypher das Muster beschrieben, das abgeglichen werden muss (Zeile 18-23) und mit den benötigten Werten (Zeile 25-29) zu

einer Abfrage im JSON-Format zusammengefasst (Zeile 31-34). Diese kann anschließend in Zeile 42 in der `ajax()`-Funktion übergeben werden. Die HTTP-Methode ist vom Typ `POST` (Zeile 37) und die in 4.1.2 bereits erwähnte Angabe des Datenformats erfolgt in den Zeilen 38 und 41. Innerhalb der `success()`-Funktion werden die vom Neo4j-Server zurückgelieferten Ergebnisdaten als `result`-Parameter an die Funktion `$.gevent.publish()` weitergereicht (Zeile 44-46). Diese Funktion stellt einen Mechanismus bereit, der die Daten für andere Module verfügbar macht und in Abschnitt 4.2.3 *Custom Events* genauer beschrieben wird.

Listing 4.2: Das JavaScript–Modul app.data.js (vereinfacht)

```
1  app.data = (function () {
2    var
3          URL_TO_NEO4J_DB = "http://localhost:7474/db/data",
4          CYPHER_ENDPOINT = "/cypher",
5          lookupAllUserNodes,
6          [...]
7          initModule
8          ;
9    [...]
10   lookupAllUserNodes = function (username, trainingID) {
11         var
12           cql,
13           query,
14           params
15           ;
16
17         /* Cypher Query Language */
18         cql = [
19           'START training=node:node_auto_index(type={type})',
20           'MATCH (user)-[r:ATTENDS_CCT]->(training)',
21           'WHERE training.id={id} AND NOT(user.name={name})',
22           'RETURN user.name'
23         ].join('\n');
24
25         params = {
26           "type"  : "cct",
27           "id"    : trainingID,
28           "name"  : username
29         };
30
31         query = {
32           "query"   : cql,
```

```
33            "params"  : params
34          };
35
36          $.ajax({
37            type            : "POST",
38            accepts         : "application/json",
39            url             : URL_TO_NEO4J_DB + CYPHER_ENDPOINT,
40            dataType        : "json",
41            contentType     : "application/json",
42            data            : JSON.stringify(query),
43            [...]
44            success         : function (result, textStatus, xhr) {
45                $.gevent.publish('allUserSelection', result);
46            },
47            error           : function (xhr, textStatus) {
48                console.log("POST error: " + textStatus);
49            }
50          });
51      };
52
53      [...]
54
55      initModule = function () {
56      };
57
58      /* Return the public methods */
59      return {
60          [...]
61          lookupAllUserNodes  : lookupAllUserNodes,
62          initModule          : initModule
63      };
64  }());
```

Ebenfalls zum Umfang des Prototyps gehören kleinere Module (`app.util.js`), die nützliche Funktionen bereitstellen, auf die alle anderen Module zugreifen können oder provisorische Module (`app.dummy.js`), die während der Entwicklungsphase dazu dienen, Beispieldaten aus der Datei `data.js` zu laden.

Abbildung 4.6 soll den beschriebenen Aufbau und die Kommunikationswege der JavaScript-Module verdeutlichen. Die für die Benutzeroberfläche verantwortlichen Module haben – bis auf `app.shell.js` – keine Kenntnis voneinander und interagieren lediglich mit dem Modul `app.model.js`, das wiederum mit `app.data.js` kommuniziert.

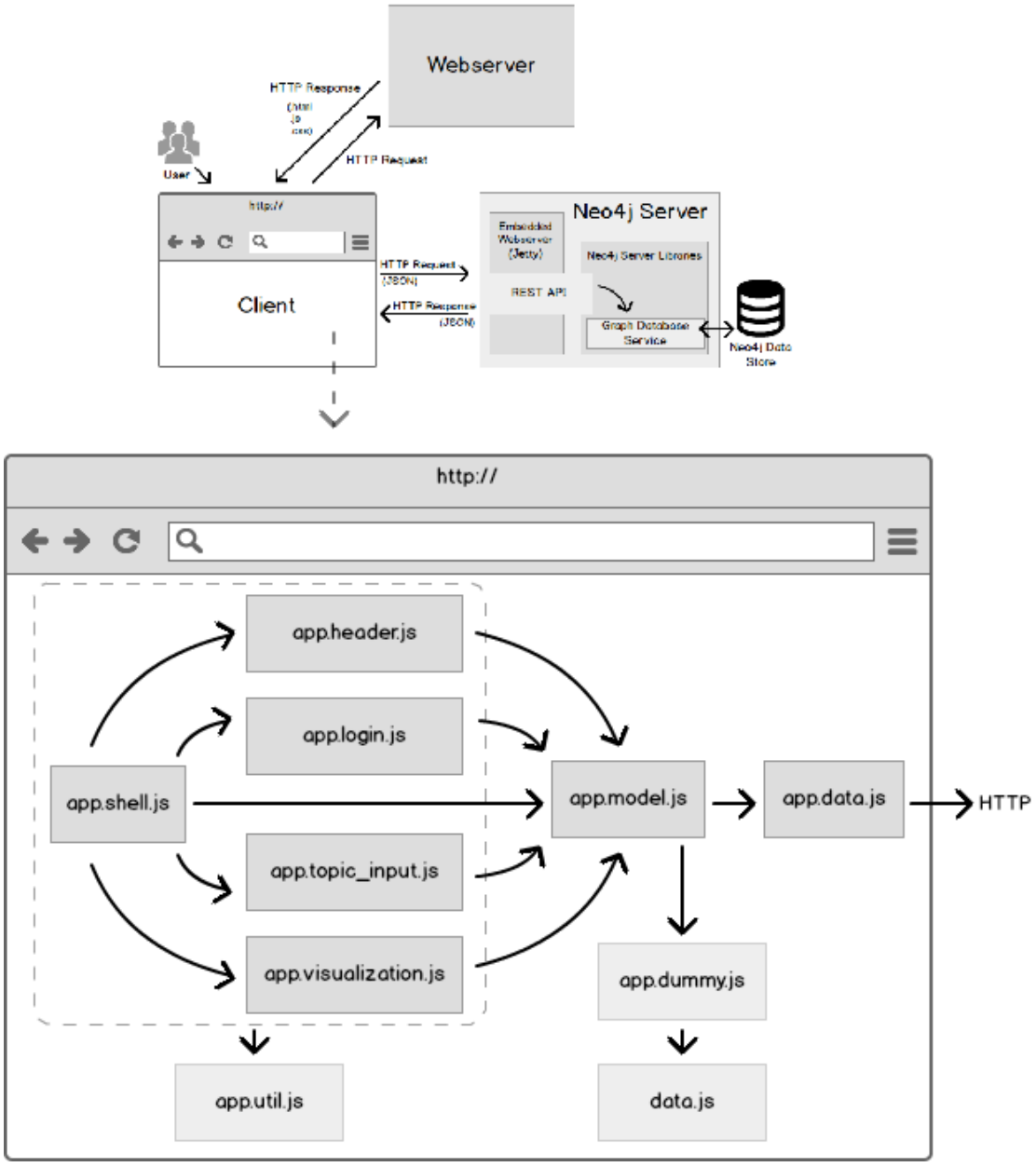


Abbildung 4.6: Client-Applikationsarchitektur: zu erkennen ist die lose Kopplung der Module und anhand der Pfeilrichtung wird deutlich wie die Interaktion erfolgt, d.h. welche Module auf die öffentlichen Schnittstellen anderer Module zugreifen können [vgl. Abbildung aus MP13, S. 140].

4.2 Architektur des Client-Codes mithilfe von JavaScript Entwurfsmustern

Entwurfsmuster beschreiben in der Softwareentwicklung Vorgehensweisen, die sich zur Lösung häufig auftretender Probleme bewährt haben. So ist die in Abschnitt 4.1.3 erwähnte MVC-Struktur als ein mögliches Muster zu verstehen, wie man die Aufgabe lösen kann, eine Anwendung und die Interaktion ihrer Komponenten zu organisieren. Grundsätzlich sind Entwurfsmuster programmiersprachenunabhängig, eignen sich aber nicht immer für den Einsatz in einer bestimmten Sprache oder sind sogar unnötig. Für JavaScript lassen sich hingegen auch Muster oder sogenannte *best practices* benennen, die spezifische Eigenheiten der Sprache adressieren und gleichzeitig dabei helfen, gängige Fehler und Probleme bei der Programmierung in JavaScript zu vermeiden [vgl. Ste10, S. 1-2]. Nachfolgend werden die *JavaScript Patterns* beschrieben, die bei der Umsetzung der Web-Applikation zum Einsatz kommen.

4.2.1 Namespace Pattern

Damit auf der globalen Ebene des Webbrowsers die Anzahl der globalen Variablen und Objekte möglichst gering bleibt und es nicht zu Namenskonflikten innerhalb des eigenen JavaScript-Quellcodes oder mit externen JavaScript-Bibliotheken[27] kommt, wird das Modul `app.js` als einziges globales Objekt der Web-Applikation zur Verfügung gestellt und definiert den Namensraum `app` und somit einen Geltungsbereich, unter dem auf Variablen und Funktionen dieses Objekts als dessen Eigenschaften zugegriffen werden kann. Dieser Geltungsbereich wird mit `app.shell`, `app.header`, `app.model` etc. weiter untergliedert [vgl Ste10, S. 87-88; MP13, S. 535-536]. Mit dem Namespace Pattern lässt sich allerdings nicht kontrollieren, auf *welche* Eigenschaften global zugegriffen werden darf. Hierzu dient das im nächsten Abschnitt erläuterte *Revealing Module Pattern*.

4.2.2 Grundaufbau der JavaScript-Module nach dem Revealing Module Pattern

Der modulare Aufbau der Applikationsarchitekur und die Vorteile dieser Strategie wurden bereits in Abschnitt 4.1.3 dargestellt. Nun soll ergänzend das Muster vorgestellt werden, nach dem die Programmierung der JavaScript-Module erfolgt. Das Namespace Pattern ist dabei

[27] Die JavaScript-Bibliothek jQuery hat beispielsweise eine `noConflict()`-Methode, da viele andere JavaScript-Bibliotheken das `$`-Symbol wie jQuery selbst auch als Name für eine globale Funktion oder Variable benutzen. Siehe hierzu *http://api.jquery.com/jQuery.noConflict/* (besucht am 07.09.2013).

Bestandteil des Revealing Module Pattern. Ein weiteres Kennzeichen eines Module Pattern ist die Verwendung von *Immediate Functions*. Das sind anonyme Funktionen mit einer speziellen Syntax, die sich beim Laden der Seite sofort selbst ausführen. Eine Immediate Function ist daran erkennbar, dass sich am Ende der Funktion zusätzlich eine öffnende und schließende runde Klammer befindet und die gesamte Funktion in runde Klammern gesetzt wird (siehe Listing 4.3 Zeile 2 und Zeile 61). Alles was innerhalb dieser sich-selbst-ausführenden anonymen Funktion definiert wird, befindet sich im Geltungsbereich dieser Funktion. Wie bei anderen JavaScript-Funktionen auch, kann ein Rückgabewert bestimmt und dieser einer Variablen zugewiesen werden. Bei Verwendung des Revealing Module Pattern sind alle Variablen und Funktionen innerhalb der Immediate Function zunächst privat und es wird erst in der Rückgabe bestimmt, was öffentlich gemacht wird – *revealing* bedeutet auf deutsch *enthüllend*, *aufdeckend*. Letztendlich wird ein Objekt zurückgegeben und dies ist das eigentliche Modul mit seinen öffentlichen Methoden, seiner öffentlichen Schnittstelle [vgl. Ste10, S. 69-72, 97-99].

Die innere Strukturierung der browserspezifischen Module erfolgt immer nach einem gleichen Schema, sodass der Aufbau modulübergreifend konsistent ist. Dies ist eine übliche Praxis bei der Programmierung von JavaScript-Modulen und es gibt verschiedene Ansätze, diese Struktur zu gestalten, wobei die Vorgehensweise nicht zuletzt auf Erfahrung beruht, die aus der Arbeit an Projekten gewonnen wird.[28] Der für die Umsetzung des Prototyps verwendete und in Listing 4.3 ausschnitthaft veranschaulichte Aufbau lehnt an das „module template“ aus dem Buch „Single Page Web Applications“ an [vgl. MP13, S. 546].

Listing 4.3: Innere Modulstruktur nach dem Revealing Module Pattern (vereinfacht)

```
1  // Creating the namespace of this module: app.visualization
2  app.visualization = (function () {
3    var
4      // A configuration map with values that do not change after module ↩
           initialization
5      configMap = {
6        rendered_html : app.util.renderTemplate('app.visualization')
7      },
8      // A map that will hold values which change dynamically
9      stateMap = {},
10     // A map that will hold cached jQuery selection objects
11     jqueryMap = {},
12     setJqueryMap,
13     bindUIActions,
14     subscribeEvents,
```

[28] http://www.impressivewebs.com/my-current-javascript-design-pattern/ (besucht am 27.08.2013

```
15      [...]
16      initModule;
```

Am Anfang der Immediate Function erfolgt immer die Deklaration aller privaten Variablen (Zeile 3-16). Zeile 6 zeigt dabei exemplarisch, wie das zu diesem Modul gehörende Handlebars-Template zugewiesen wird.

Da in den Modulen, die für Benutzerinteraktionen zuständig sind, oft auf die gleichen Elemente (Buttons, Eingabe-Felder, etc.) zugegriffen werden muss, bietet es sich an, diese mittels jQuery-Selektoren in einem Objekt zu cachen (Zeile 19-28). Hier wird auch die Konvention deutlich, das Präfix `$` für die Namen der Variablen von jQuery-Objekten zu verwenden [vgl. MP13, S. 527].

```
17
18    // Populates the jqueryMap with jQuery selection objects.
19    setJqueryMap = function () {
20      var $templateContainer;
21      $templateContainer = configMap.$template_container;
22      jqueryMap = {
23          $template_container : $templateContainer,
24          $welcomePane        : $templateContainer.find('#welcome'),
25          $pairTab            : $templateContainer.find('a[href="#pair"][
                data-toggle="tab"]'),
26        [...]
27      };
28    };
```

Ebenfalls nützlich sind die Funktionen in Zeile 31-34 und Zeile 37-40, um die Bindung zwischen den Seitenelementen und den Events herzustellen. Dies strukturiert den Quellcode und zusätzlich erfolgt eine Trennung zwischen Browser-Events und globalen Custom Events (siehe 4.2.3).

```
29
30    // Convenient function for binding user interface events
31    bindUIActions = function () {
32      jqueryMap.$pairTab.on('shown', fetchUserSelection);
33      [...]
34    };
35
36    // Convenient function for subscribing to global custom events.
37    subscribeEvents = function () {
```

```
38      $.gevent.subscribe(jqueryMap.$pairPane, 'allUserSelection', ↩
            addSelectOptions);
39    [...]
40    };
41
42  [...]
```

In obigem Beispiel fehlen die Event-Handler sowie die modulspezifischen Funktionen, da mit diesem exemplarischen Ausschnitt vornehmlich gezeigt werden soll, nach welcher Grundstruktur die Module implementiert sind.

```
43    // Configures and initializes the module.
44    initModule = function ($container) {
45      // Find the target container within which the template is going to be ↩
            rendered.
46      configMap.$template_container = $container;
47      // Insert the rendered template.
48      configMap.$template_container.html(configMap.rendered_html);
49      // After the template is in place, jQuery selection objects can be ↩
            set
50      setJqueryMap();
51      // and the module can be bound to browser events
52      bindUIActions();
53      // and to global custom events.
54      subscribeEvents();
55    };
56
57    // Return the public method **initModule**
58    return {
59      initModule : initModule
60    };
61  }());
```

Die `initModule()`-Funktion steht immer ganz am Ende und diese wird dann als einzige Methode durch die `return`-Anweisung öffentlich gemacht (im Unterschied zu den anderen Modulen wie `app.model` oder `app.data`, die weitere öffentliche Methoden zur Verfügung stellen müssen). Damit ist es nun möglich, die Module von außen zu initialisieren. Dies geschieht durch die `$(document).ready()`-Funktion (ersichtlich in Listing 4.1). Hier beginnt durch den Aufruf von `app.initModule("#app")` eine Initialisierungs-Kaskade, da das Hauptmodul `app` in seiner `initModule()`-Funktion die Initialisierung der Module `app.model`, `app.data` und `app.shell` veranlasst. Letzteres wiederum führt in seiner `initModule()`-Funktion die

Initialisierung der anderen Module fort [vgl. MP13, S. 161-163].

4.2.3 Custom Events

Custom Events beschreiben einen Mechanismus, der es möglich macht, auf ein bestimmtes Ereignis innerhalb der Anwendung reagieren zu können, das nicht durch eine Interaktion mit der Benutzeroberfläche (wie z.B. ein Mausklick) ausgelöst wird. Eine alternative Bezeichnung für dieses JavaScript Pattern ist *Publish/Subscribe*, da es einen *publisher* gibt, der das eingetretene Ereignis bekannt macht und einen oder mehrere *subscriber*, die sich für den Erhalt dieser Bekanntmachung anmelden. Dabei wissen beide Parteien nichts voneinander, da es für den Publisher unerheblich ist, wer die Nachricht empfängt. Er sendet sie einfach global aus. Auf der anderen Seite interessieren sich die Subscriber ausschließlich für die Nachricht und nicht von wem sie kommt. Zusammen mit einer Nachricht können auch Daten übergeben werden und so haben die Empfänger die Möglichkeit, diese zu nutzen und zu verarbeiten. Die Custom Events werden in der Web-Applikation dazu genutzt, um die Daten, die das Modul `app.data` von der Datenbank empfängt, für andere Module verfügbar zu machen, ohne dass eine weitere Interaktion zwischen den Modulen erforderlich ist (siehe Listing 4.2 Zeile 45 sowie Listing 4.3 Zeile 37). Auf diese Weise kann eine lose gekoppelte Anwendung realisiert werden [vgl. Mac11, S. 25-30]. Zur Realisierung von Custom Events bzw. Publish/Subscribe kann man auf Plugins zurückgreifen. Für den Prototyp werden die Custom Events mithilfe des Plugins *jquery.event.gevent*[29] implementiert.

4.3 Anwendung der Frameworks und Bibliotheken

Nachfolgend soll der praktische Einsatz der Frameworks und Bibliotheken, mit deren Hilfe die Umsetzung der Web-Applikation erfolgt und auf die im bisherigen Text bereits mehrfach Bezug genommen wurde, anhand von Beispielen gezeigt werden (lediglich jQuery wird nicht separat aufgeführt, da sich der Einsatz dieser Bibliothek durch das ganze Projekt zieht und exemplarisch Listing 4.2 oder Listing 4.3 entnommen werden kann).

4.3.1 Bootstrap

Das Front-End/CSS-Framework Bootstrap wird in der Version 2.3.2 verwendet und hilft maßgeblich bei der Gestaltung der HTML-Seitenbereiche. Die von Bootstrap bereitgestellten

[29] vgl. *https://github.com/mmikowski/jquery.event.gevent* (besucht am 27.08.2013)

Vorlagen lassen sich mithilfe der Dokumentation gut an die eigenen Bedürfnisse anpassen, wodurch man schnell ein ansprechendes Ergebnis erreicht, das zunächst als Basis für den Prototyp dienen kann. Verwendung finden neben den Standardelementen für Auswahllisten, Buttons oder Eingabefelder auch sogenannte *Alerts* zur Anzeige von Fehlermeldungen sowie die bei der Konzeption der Benutzeroberfläche bereits berücksichtigten Layout-Möglichkeiten von Bootstrap für ausklappbare Menüs und für *togglable tabs*, also die Registerkarten-Funktionalität. Hierzu ist es notwendig, neben der `bootstrap.css`-Datei auch die `bootstrap.js`-Datei einzubinden, um die mitgelieferten JavaScript-Funktionen nutzen zu können. Ebenfalls eingebunden wird die Datei `bootstrap-responsive.css`, damit sich das Layout an die Größe des Bildschirms anpasst. Die Funktion `showAlert()` in Listing 4.4 zeigt ein Beispiel, wie bei der Implementierung die Bootstrap Alert-Meldungen eingesetzt werden. Zeile 3-6 entspricht der Vorlage von Bootstrap, es muss hier lediglich für den Text (Zeile 5) und in den CSS-Regeln bezüglich der Breite eine Modifikation erfolgen. Um die Meldung wieder auszublenden, kann man dann auf eine von Bootstrap mitgelieferte Funktion zurückgreifen, was letztendlich nur eine Zeile im Quellcode ausmacht und die Arbeit sehr erleichtert. Das entsprechende Layout der Alert-Meldung ist in Abbildung 4.7 zu sehen.

Listing 4.4: Funktion zur Anzeige einer Bootstrap Alert–Meldung

```
1 showAlert = function(){
2   jqueryMap.$alert_div.append([
3     '<div class="alert alert-danger fade in">',
4     '<button type="button" class="close" data-dismiss="alert">x</button
        >',
5     'Please, select a Trainings-ID.',
6     '</div>'
7   ].join('\n'));
8 };
```

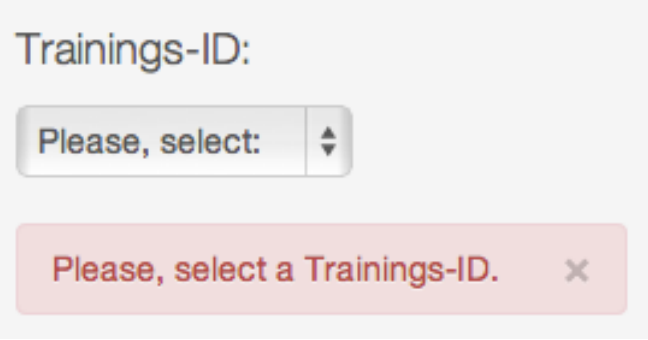


Abbildung 4.7: Anzeige einer Bootstrap Alert-Meldung (Bildschirmfoto – Ausschnitt)

4.3.2 Handlebars

Wie in Abschnitt 4.1.3 bereits beschrieben, wird die HTML-Struktur der Web-Applikation in separaten Handlebars-Dateien für den jeweiligen Seitenbereich erstellt. Der Einsatz von Handlebars (Version 1.0.rc.1) lässt sich anhand des Beispiels für den Bereich der Benutzereingabe gut veranschaulichen. Der Quellcodeausschnitt in Listing 4.5 zeigt einen kleinen Teil der HTML-Struktur, aus der das ausklappbare Menü gebildet wird. Darin eingebettet sind die Handlebars-Ausdrücke (zu erkennen an den geschweiften Klammern) mit den Variablen, an deren Stelle die Datenwerte aus Listing 4.6 eingefügt werden. Das Menü setzt sich aus einer ungeordneten Liste (Beginn Zeile 2, Ende Zeile 10) zusammen. Zur besseren Übersicht wird aus dem Inhalt des Listeneintrags (Beginn Zeile 4, Ende Zeile 7) lediglich das Label (Zeile 5) angezeigt, die Eingabefelder und Buttons, die nach dem Ausklappen sichtbar werden, sind ausgespart. Das Listenelement in Zeile 8 fungiert als optische Trennlinie zwischen den Einträgen.

Listing 4.5: Das Template app.topic_input.handlebars (vereinfacht)

```
1   [...]
2 <ul class="nav nav-list">
3   {{#each categories}}
4   <li>
5     <label class="tree-toggler" style="color:{{color}}">{{tag}}</label>
6           [...]
7   </li>
8   <li class="divider"></li> <!-- horizontal rule -->
9   {{/each}}
10 </ul>
11 [...]
```

Listing 4.6: Die Template–Daten aus der Datei data.js (vereinfacht)

```
1  var data = {
2    trainings  : [
3      {id : "cct-2013_1", type: "cct"}
4    ],
5    categories : [
6      {tag : "Profession", type : "category", color : "#1f77b4"},
7      {tag : "Education", type : "category", color : "#ff7f0e"},
8      {tag : "Family", type : "category", color : "#2ca02c"},
9      {tag : "Sexual-Identity", type : "category", color : "#d62728"},
10     {tag : "Geographical-Places", type : "category", color : "#9467bd"},
11     [...]
```

```
12    ]
13 };
```

Die Verarbeitung des Templates erfolgt über das Modul `app.util.js`. Dort wird in der Funktion `renderTemplate()` die `Handlebars.compile()`-Funktion aufgerufen. Bei der Verarbeitung des Templates wird nun durch die Handlebars `#each`-Funktionalität (Beginn Zeile 3, Ende Zeile 9) alles was sich innerhalb dieses Blockes befindet für jeden Eintrag aus dem `categories`-Array generiert und die Variablen mit den Werten ersetzt. Das Ergebnis ist in Abbildung 4.8 zu sehen. Die Abbildung zeigt den entsprechenden Bildschirmausschnitt mit dem DOM der geladene HTML-Seite, betrachtet mit den Chrome Developer Tools.

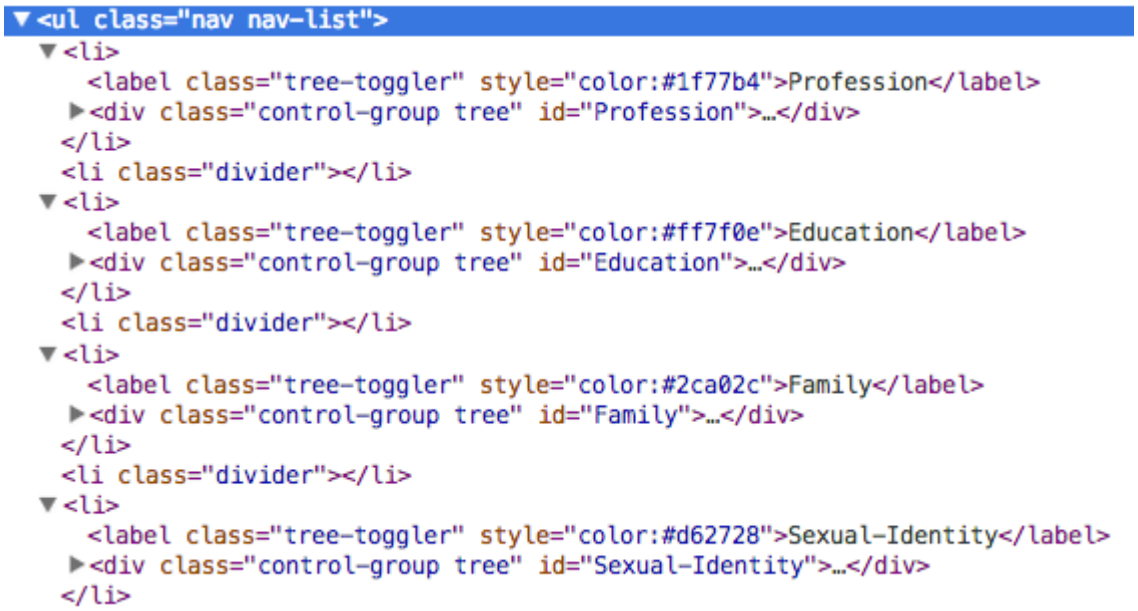


Abbildung 4.8: Handlebars Template nach der Verarbeitung (Bildschirmfoto – Ausschnitt)

4.3.3 D3

Die Visualisierungsbibliothek D3 wird in der Version 3.2.8 in das Projekt integriert. Aus Zeitgründen kann die Visualisierung nur für die Einzelsicht realisiert werden, dabei wird auf eine der in Abschnitt 2.3.2.5 bereits erwähnten Layout-Methoden von D3 zurück gegriffen, um einen einfachen Graph zu visualisieren.

Verwendet wird die `d3.layout.force()`-Methode und damit soll ein Force-Directed-Layout realisiert werden (vgl. Abschnitt 2.2.2). Hierzu sind Knoten und Kanten erforderlich, implementiert jeweils als Array bestehend aus JavaScript-Objekten. Das grundlegende Aussehen ist

in Listing 4.7 skizziert. Zeile 1 soll das `links`-Array mit den Kantenobjekten veranschaulichen, die jeweils ein Start- und Zielwert, sowie eine Farbe besitzen. Zeile 2 zeigt analog das `nodes`-Array für die Knotenobjekte.

Listing 4.7: Aufbau der Kanten–und Knotenarrays (vereinfacht)

```
1 links = [{source : <value>, target : <value>, color : <value>}, ...];
2 nodes = [{name :  <value>, color: <value> }, ...];
```

Das `links`-Array wird mit den Daten für den aktuellen Benutzer gefüllt, d.h. die Informationen aus dem Benutzerknoten sowie aus allen mit ihm verbundenen Begriffsknoten. Die dafür formulierte Cypher-Abfrage liefert ein verschachteltes Ergebnisarray mit dem Namen des Benutzers und allen von ihm eingegebenen Begriffen sowie potentiellen Unterbegriffen. Ebenfalls enthält das Ergebnisarray den Farbwert der Kategorie, in welcher der Begriff eingegeben wurde. Damit kann später die Farbe des SVG-Elements bestimmt werden. Das Ergebnisarray muss nun derart ausgewertet werden, dass der Benutzer immer eine `source` darstellt und ein Begriff immer ein `target`. Ein Begriff kann aber wiederum eine `source` sein, wenn ein Unterbegriff (`target`) eingegeben wurde. Der Quellcodeausschnitt in Listing 4.8 zeigt die Implementierung der entsprechenden Funktion, die eine verschachtelte `for`-Schleife verwendet, um die Daten wie benötigt in das `links`-Array zu speichern.

Listing 4.8: Aufbereitung der Daten für das D3 Force Layout (vereinfacht)

```
1  [...]
2  // The data returned from the database is an array and
3  // always structured in the following manner:
4  // [[username, topic1, [optional subtopics], categorycolor], [...], ...]
5  dataToSingleView = function (event, db_data) {
6   var i = 0,
7       j = 1,
8       links = [],
9       data = db_data.data;
10
11  for (i; i < data.length; i++) {
12    links.push({source : data[i][0], target : data[i][1], color : data[i
        ][3]});
13    if (data[i][2].length > 0) {
14      links.push({source : data[i][1], target : data[i][2][0], color :
          data[i][3]});
15      for (j; j < data[i][2].length; j++) {
16        links.push({source : data[i][2][j - 1], target : data[i][2][j],
            color : data[i][3]});
```

```
17        }
18      }
19    }
20  stateMap.singleViewData = links;
21  [...]
22 };
23 [...]
```

Da mit diesem Array nun die Verbindungen (Kanten, Relationen) zwischen den Knoten verfügbar sind, können daraus wiederum die Knoten erzeugt werden. Es soll jedoch an dieser Stelle lediglich noch auf die `d3.layout.force`-Methode eingegangen werden, zu sehen in dem Quellcodeausschnitt in Listing 4.9.

Listing 4.9: Die d3.layout.force()–Methode (vereinfacht)

```
1  [...]
2  width = 760;
3  height = 450;
4
5  force = d3.layout.force()
6    .nodes(d3.values(nodes))
7    .links(links)
8    .size([width, height])
9    .linkDistance(100)
10   .charge(-400)
11   .on("tick", tick)
12   .start();
13 [...]
```

D3 verwendet wie auch jQuery die „chain syntax“ [Mur13, S. 69], um Methoden aneinander zu ketten und so mehrere Aktionen in knapper Schreibweise durchzuführen. In Zeile 6 werden die Knoten, in Zeile 7 die Kanten angegeben, die genutzt werden sollen. Die Größe des Bereichs, der dem Layout zur Verfügung steht, wird in Zeile 8 bestimmt und wirkt sich auf zwei Aspekte des Force-Directed-Layouts aus: das Gravitationszentrum und die zufällige Anfangsposition [vgl. Bos13]. Mit diesen drei Methoden ist bereits ein vorgegebenes D3 Force Layout möglich. Es lassen sich aber viele Anpassungen vornehmen, beispielsweise eine Änderung der Kantenlänge (Zeile 9) oder für den Wert der „elektrostatischen Kraft“ (Zeile 10) der Simulation. Ein negativer Wert bedeutet eine Abstoßung der Knoten und sollte für ein Graph-Layout genutzt werden [vgl. Mur13, S. 210–215; Bos13]. In Zeile 11 wird ein Event-Listener für das „`tick`“-Event registriert und mit dem Start der Simulation in Zeile 12 wird dieses abgesendet. Die `tick`-Funktion lässt die Simulation voran schreiten („ticken“)

und die `x/y`-Positionen, die D3 zuvor im Hintergrund für jedes JavaScript-Objekt aus dem `links`- bzw. `nodes`-Array berechnet hat, werden mit jedem „tick“ angepasst [vgl. Mur13, S. 210–215; Bos13]. Diese Anpassung wird auf die nun im DOM sichtbaren SVG-Elemente für die Kanten (`<line/>`) und Knoten (`<circle/>`) angewendet, deren Erzeugung hier ausgespart wird. Die Implementierung erfolgt anhand eines Online-Code-Beispiels von Michael Bostock,[30] geringfügige Modifikationen betreffen die Farbwerte der Knoten, die nicht statisch angegeben sondern aus dem Array abgefragt werden und die Wahl eines größeren Wertes für den Radius des Benutzerknoten.

[30] vgl. *http://bl.ocks.org/mbostock/2706022* (besucht am 07.08.2013)

Kapitel 5

Ergebnisse und Auswertung

Dieses Kapitel dient der Demonstration des Prototypen und präsentiert die Ergebnisse, die bis zum Zeitpunkt der Abgabe dieser Bachelorarbeit erzielt werden konnten. Dabei wird auch darauf eingegangen, wie gut die Lösungsansätze funktionieren und ob die entwickelte Applikations-Architektur die Anforderungen erfüllen kann, damit sich der Prototyp zu einer produktiv einsetzbaren Web-Applikation erweitern lässt.

5.1 Demonstration des Prototyps

Registrierung und Anmeldung

Der Dialog zur Registrierung konnte erfolgreich umgesetzt werden. Wie bereits in Abbildung 4.5 veranschaulicht, findet dabei auch eine Prüfung statt, ob alle erforderlichen Felder ausgefüllt sind. Bisher noch nicht erfolgreich umgesetzt ist eine Prüfung, ob der gewählte Benutzername bereits existiert und ebenfalls fehlt bislang eine direkte Validierung der eingegebenen Emailadresse. Für den Login-Dialog wurde die Anmeldung so realisiert, dass diese mit dem Benutzernamen oder mit der Emailadresse und dem gewählten Passwort möglich ist, auch hier erhält der Benutzer eine Fehlermeldung, wenn die Daten nicht korrekt sind. Dies ist in Abbildung 5.1 zu erkennen. Die dort ersichtliche „Passwort vergessen?“-Funktion ist allerdings für den Prototyp noch nicht implementiert. Nach einer erfolgreichen Registrierung oder Anmeldung wird der Dialog wie im Entwurf vorgesehen mit einer „Slide up“-Animation aus- und die Oberfläche der Anwendung eingeblendet.

Abbildung 5.2 zeigt die Benutzeroberfläche direkt nach erfolgreicher Anmeldung. Der Login-Status im Header-Bereich hat sich geändert und zeigt nun den Namen des angemeldeten Benutzers sowie den Logout-Button an. Im Hauptbereich wurde die Idee aus der Konzeption noch erweitert, sodass es nun insgesamt vier Tabs gibt. Der „Welcome“-Tab ist standardmäßig

aktiv und soll dazu dienen, den Benutzer zu begrüßen und Informationen zu Sinn, Zweck und Funktionsweise bereitzustellen.

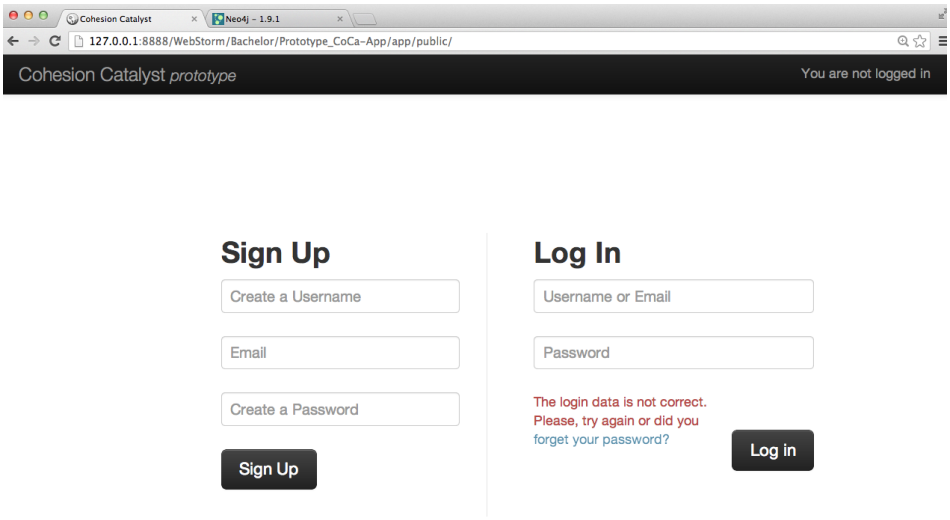


Abbildung 5.1: Das Ergebnis im Webbrowser für den Dialog zur Registrierung bzw. Anmeldung, hier zur Veranschaulichung mit einer Fehlermeldung.

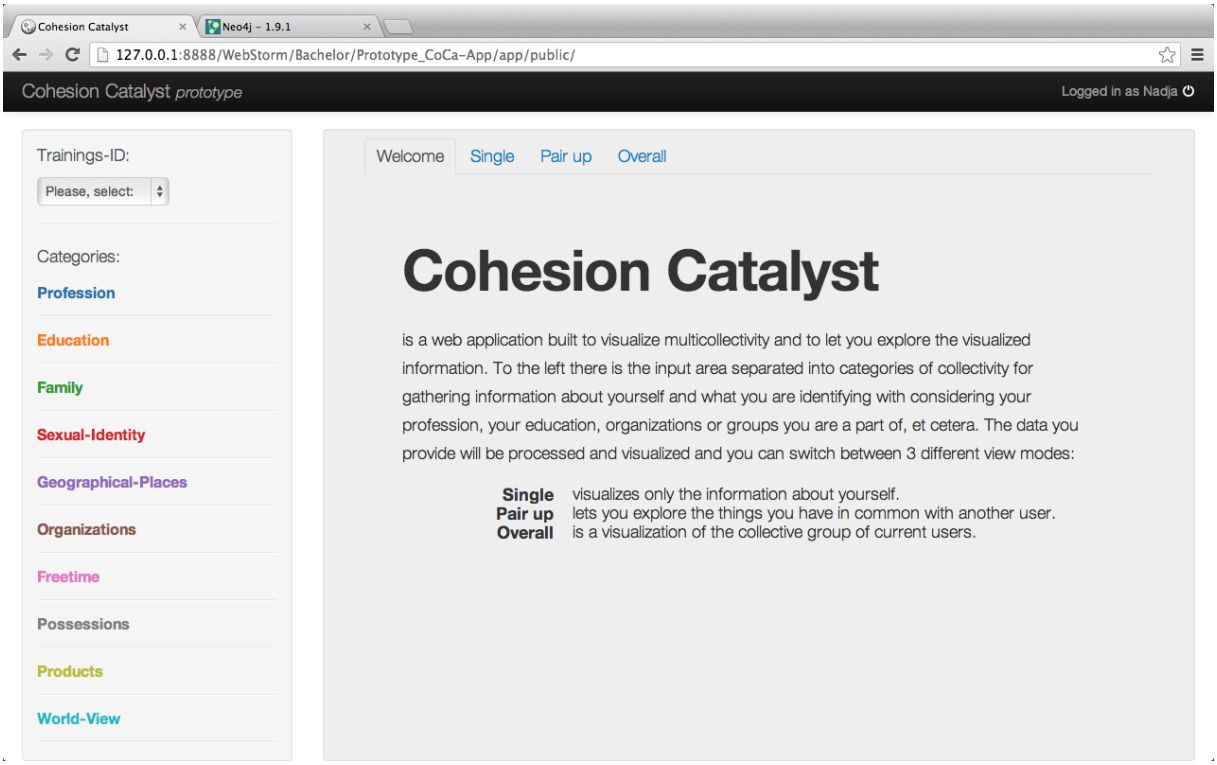


Abbildung 5.2: Die Benutzeroberfläche des Prototyps unmittelbar nach dem Login, oben rechts im Header-Bereich ist der aktualisierte Login-Status im Vergleich zu Abbildung 5.1 zu erkennen.

Benutzereingabe

Abbildung 5.3 zeigt das Ergebnis für den Seitenbereich und die Benutzereingabe und wie dort die Eingabefelder bei Bedarf hinzugefügt bzw. auch wieder entfernt werden können. Die Schriftfarbe der jeweiligen Kategorien soll als visuelle Codierung die spätere Exploration der Visualisierung unterstützen. Hier hat sich bereits während der Entwicklung gezeigt, dass der aktuelle Lösungsansatz sich recht schnell als unpraktisch erweist, wenn viele Begriffsverfeinerungen vorgenommen werden und dadurch die Eingabefelder immer mehr einrücken. Zwar lässt sich dieser Bereich in solch einem Fall horizontal scrollen, allerdings erscheint dadurch die Benutzerfreundlichkeit eingeschränkt und daher sollte hier über eine Optimierung nachgedacht werden. Gleiches gilt für die Realisierung eines aussagekräftigen Feedbacks an den Benutzer bei der Interaktion mit der Web-Applikation. Dies ist noch nicht ausreichend implementiert. Insbesondere Fehlermeldungen, z.B. wenn es zu Fehlern mit der Anfrage an die Datenbank kommt, sind aktuell nur über die Console der Developer-Tools erkennbar.

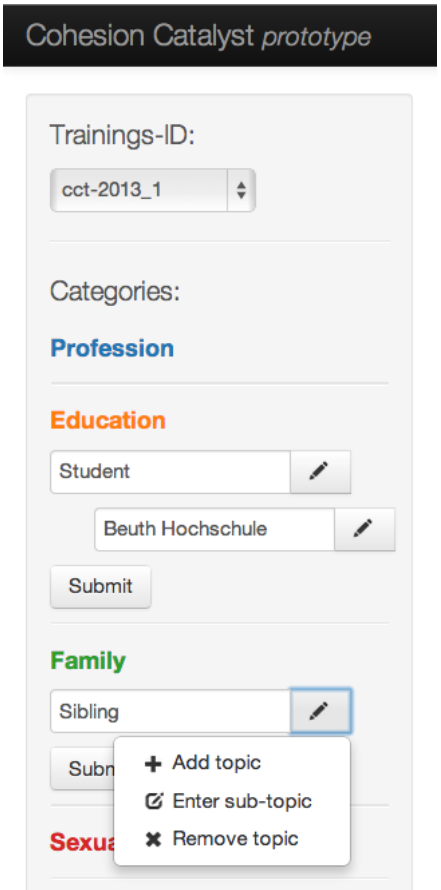
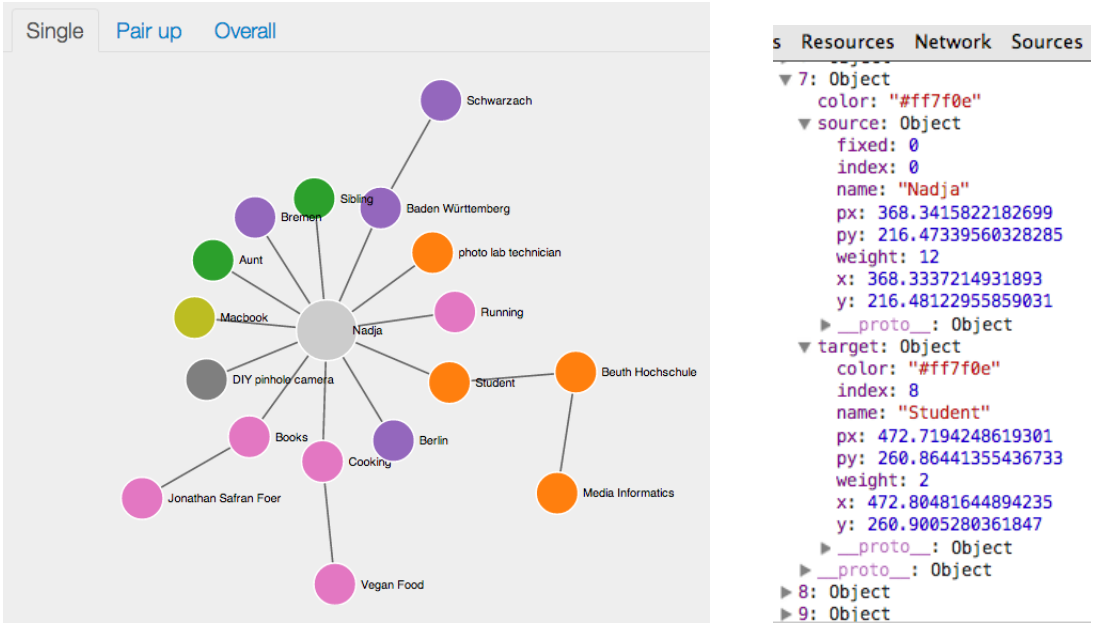


Abbildung 5.3: Der Seitenbereich für die Benutzereingaben mit ausgeklappter Kategorie. Oberhalb der Kategorien ist das Auswahllistenelement für die Trainings-ID zu sehen.

Visualisierung und interaktive Exploration

Die in Abbildung 5.4(a) ersichtliche Visualisierung konnte wie in Abschnitt 4.3.3 erläutert aus Zeitgründen lediglich für die Einzelsicht realisiert werden. Das Ergebnis stellt daher einen ersten beispielhaften Versuch dar, die Visualisierungsbibliothek D3 einzusetzen und Möglichkeiten zur Erweiterung bieten sich auf vielerlei Art. Hierzu kann der entwickelte Prototyp der Web-Applikation nun herangezogen werden.



(a) Automatisch erzeugtes D3 Force Layout **(b)** Berechnete x/y-Werte

Abbildung 5.4: (a) zeigt das mit D3 realisierte „Force Directed Layout“ zu den eingegebenen Begriffen eines Benutzers. In (b) ist ein Ausschnitt des DOM dargestellt, nachdem die Visualisierung generiert wurde und soll die zusätzlich von D3 berechneten Eigenschaften (wie z.B. x- und y-Position) veranschaulichen.

Die Abfrage der Datenbank zur Visualisierung der 1:1 Sicht liefert die Namen der anderen Benutzer und diese werden dem Auswahllistenelement unter dem „Pair up“-Tab (siehe Abbildung 5.5) hinzugefügt.

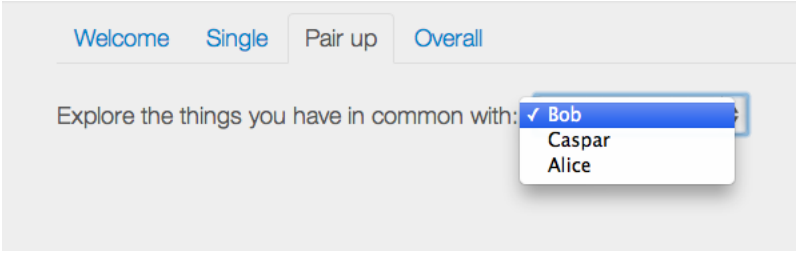


Abbildung 5.5: Auswahllistenelement für 1:1 Visualisierung.

5.2 Überprüfung der Anforderungen

Tabelle 5.1 soll einen Überblick über die in Abschnitt 3.2 aufgestellten wesentlichen Anforderungen und den Status der Erfüllung geben, soweit dies im Rahmen der Prototyp-Entwicklung gelungen ist. Bei den erfüllten Anforderungen kommt es nun darauf an, diese weiter zu testen und zu optimieren, insbesondere hinsichtlich einer sorgfältigen Fehlerbehandlung. Die nur teilweise und die nicht erfüllten Anforderungen können im Zuge einer Weiterentwicklung zusammen mit den ebenfalls offenen Wunschkriterien in Angriff genommen werden.

Tabelle 5.1: Übersicht der erfüllten Anforderungen

Anforderung	erfüllt	teilweise erfüllt	nicht erfüllt
Registrierung und Anmeldung	✓		
Moderator-Funktionalität			•
Datenerfassung	✓		
Kategorien und wahlfreie Eingabe	✓		
lose Begriffseingabe	✓		
Verfeinerung der Begriffe	✓		
Löschen von Begriffen		•	
Aussagekräftige Visualisierung		•	
Dynamische Visualisierung			•
Verschiedene Betrachtungsmodi		•	
Englische Sprachunterstützung	✓		
Intuitive Bedienbarkeit	✓		

Kapitel 6

Fazit und Ausblick

Die Aufgabe im Rahmen dieser Bachelorarbeit bestand darin, ein Konzept für eine Web-Applikation zu entwickeln, die in interkulturellen Trainings eingesetzt werden kann, um gemeinsame Gruppenzugehörigkeiten der Teilnehmenden durch eine geeignete Visualisierung dieser Multikollektivität interaktiv erforschbar zu machen. Aus diesem konzeptionellen Entwurf sollte ein erster Prototyp der gewünschten Web-Applikation resultieren, mit dem geprüft werden kann, ob bzw. mit welchen Mitteln sich die Ideen und Lösungsansätze in die Realität umsetzen lassen.

Sowohl im Bereich der Informationsvisualisierung als auch in der modernen Webentwicklung rund um JavaScript und HTML5 bieten sich mannigfaltige Möglichkeiten zur Lösung einer bestimmten Aufgabe und unter diesem Aspekt stellte die Suche nach den geeigneten Werkzeugen für die Prototyp-Entwicklung eine Herausforderung dar. Gleichzeitig bietet gerade ein Prototyp die Chance, mit verschiedenen Ansätzen zu experimentieren, um das für die Problemstellung passende Verfahren herauszufinden. Dennoch war es hilfreich, sich frühzeitig auf bestimmte Technologien und Vorgehensweisen festzulegen. Die Recherchen zu den diese Bachelorarbeit umfassenden Themen führten u.a. zu der REST API der Graphdatenbank Neo4j und der JavaScript-Visualisierungsbibliothek D3 und ermöglichten die vollständige Prototyp-Entwicklung in JavaScript. Die Entscheidung für eine clientseitige Web-Applikation bestimmte maßgeblich die Konzeption einer modularen Applikationsarchitektur und das Front-End/CSS-Framework Bootstrap hat bei der Gestaltung der Benutzeroberfläche inspiriert und bei deren Realisierung unterstützt.

Die Programmierung in JavaScript erfolgte nach spezifischen JavaScript Coding Patterns und empfohlenen Vorgehensweisen für die Entwicklung einer JavaScript Web-Applikation. So konnten viele typische Fehler bei der Programmierung vermieden werden. Gleichzeitig ging es besonders auch darum, von vornherein einen strukturierten Aufbau der JavaScript-Module zu verfolgen. Hierbei half z.B. auch die Template Engine Handlebars. Ziel war es, ein

System aus unabhängigen JavaScript-Modulen zu entwerfen (den Prototypen) und damit eine Plattform bereitzustellen, die sich zum Testen und Experimentieren von Visualisierungs- und Interaktionstechniken eignet und anhand der Ergebnisse mit Blick auf die Weiterführung des Forschungsprojekts zu einem real einsetzbaren Produkt erweiterbar ist.

Durch die bisherige Beschäftigung mit den verschiedenen Frameworks und Bibliotheken entstanden viele neue Ideen. Dies betrifft beispielsweise die Benutzeroberfläche, bei deren Gestaltung das Front-End/CSS Framework Bootstrap bislang lediglich als Grundgerüst genutzt und nicht ausgeschöpft wird. Daher öffnen sich hier viele Optionen, den Prototyp auszubauen, nicht zuletzt auch in Bezug auf die Erarbeitung eines individuellen Designs für die Benutzeroberfläche.

Aus zeitlichen Gründen konnten nicht alle aufgestellten Projektziele vollständig erfüllt werden, beispielsweise fehlt die Umsetzung der Moderator-Funktionalität oder weitere Visualisierungen und Interaktionsmöglichkeiten. Insbesondere die letztgenannten Punkte sind wesentlich und charakteristisch für die gewünschte Web-Applikation und dies wurde nur ansatzweise erreicht. Die Einarbeitung in die Graphdatenbank Neo4j hat viel Gewicht und Zeit eingenommen, was zu Beginn der Arbeit nicht vorhersehbar war und darunter hat z.B. die Umsetzung der Visualisierung gelitten. Dies bedeutet aber auch, dass mit Neo4j eine gute Basis geschaffen wurde, auf der sich nun aufbauen lässt. Zukünftige Versionen werden, davon ist bei der Agilität des Neo4j-Teams auszugehen, Neo4j weiter optimieren und somit auch die Einsatzmöglichkeiten innerhalb des Projekts.

Die im Rahmen dieser Arbeit erfolgte thematische Auseinandersetzung mit den Konzepten und Methoden der Informationsvisualisierung und speziell der Graphvisualisierung lieferte wertvolle Erkenntnisse und Anregungen, wie sich Gemeinsamkeiten und Beziehungsstrukturen in Netzwerken aussagekräftig visualisieren lassen. Mit dem entwickelten Prototyp ist nun eine Ausgangsbasis geschaffen, verschiedene Visualisierungstechniken und Interaktionsmöglichkeiten zu erproben sowie zu vergleichen und so liefern die Ergebnisse dieser Bachelorarbeit einen ersten Beitrag für die Umsetzung einer Web-Applikation zur visuellen Exploration von Multikollektivität.

Quellenverzeichnis

Literatur

[BOH10] Michael Bostock, Vadim Ogievetsky und Jeffrey Heer. »A Tour through the Visualization Zoo«. In: *Queue* 8.5 (Mai 2010). URL: *http://doi.acm.org/10.1145/1794514.1805128*.

[BOH11] Michael Bostock, Vadim Ogievetsky und Jeffrey Heer. »D3: Data-Driven Documents«. In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011). URL: *http://vis.stanford.edu/papers/d3*.

[Cro08] Douglas Crockford. *JavaScript: The Good Parts*. 1. Aufl. Sebastopol, CA: O'Reilly Media, 2008.

[Edl+11] Stefan Edlich u. a. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. 2. Aufl. München: Hanser Fachbuchverlag, 2011.

[Fek+08] Jean-Daniel Fekete u. a. »The Value of Information Visualization«. In: *Information Visualization: Human-Centered Issues and Perspectives*. Hrsg. von Andreas Kerren u. a. Berlin [u.a.]: Springer, 2008.

[HMM00] Ivan Herman, Guy Melançon und M. Scott Marshall. »Graph Visualization and Navigation in Information Visualization: A Survey«. In: *IEEE Transactions on Visualization and Computer Graphics* 6.1 (Jan. 2000), S. 24–43.

[MG10] Stefan Münz und Clemens Gull. *HTML5 Handbuch*. Poing: Franzis Verlag, 2010.

[MP13] Michael S. Mikowski und Josh C. Powell. *Single Page Web Applications*. Manning Publications, 2013. MEAP Version 8.

[Mac11] Alex MacCaw. *JavaScript Web Applications*. 1. Aufl. Sebastopol, CA: O'Reilly Media, 2011.

[Mur13] Scott Murray. *Interactive Data Visualization for the Web*. 1. Aufl. Sebastopol, CA: O'Reilly Media, 2013.

[PVW13] Jonas Partner, Aleksa Vukotic und Nicki Watt. *Neo4j in Action*. Manning Publications, 2013. MEAP Version 8.

[RWE13] Ian Robinson, Jim Webber und Emil Eifrem. *Graph Databases*. 1. Aufl. Sebastopol, CA: O'Reilly Media, 2013.

[Rat12] Stefanie Rathje. »Multikollektivität. Wissenschaftlicher Vortrag«. Wissenschaftliche Fachtagung "Kultur und Kollektiv", Universität Passau, 25.5.2012 - 26.5.2012. Mai 2012.

[Rat13] Stefanie Rathje. »Multikollektivität. Schlüsselbegriff der modernen Kulturwissenschaften«. In: *Kultur und Kollektiv. Festschrift für den Kulturwissenschaftler Prof. Dr. Klaus-Peter Hansen*. Hrsg. von Stephan Wolting und Elias Jammal. 2013. Erscheint in Wissenschaftsverlag Berlin, 2013.

[SM04] Heidrun Schumann und Wolfgang Müller. »Informationsvisualisierung: Methoden und Perspektiven«. In: *it - Information Technology* 3 (2004), S. 135–141. URL: *http://www.informatik.uni-rostock.de/~schumann/papers/2004+/it0403_schumann_mueller.pdf*.

[SM11] David Sawyer McFarland. *JavaScript & jQuery: The Missing Manual*. 2. Aufl. Sebastopol, CA: O'Reilly Media, 2011.

[Shn03] Ben Shneiderman. »The eyes have it: A task by data type taxonomy for information visualizations«. In: *The Craft of Information Visualization*. Hrsg. von Benjamin B. Bederson und Ben Shneiderman. San Francisco, CA: Morgan Kaufmann Publishers, 2003, S. 364–371.

[Spe07] Robert Spence. *Information Visualization: Design for Interaction*. 2. Aufl. Essex: Pearson/Prentice Hall, 2007.

[Sta13] Torsten Stapelkamp. *Informationsvisualisierung: Web - Print - Signaletik. Erfolgreiches Informationsdesign: Leitsysteme, Wissensvermittlung und Informationsarchitektur*. Berlin [u.a.]: Springer, 2013.

[Ste10] Stoyan Stefanov. *JavaScript Patterns*. 1. Aufl. Sebastopol, CA: O'Reilly Media, 2010.

Internequellen

[Bos13] Michael Bostock. *Force Layout*. 2013. URL: *https://github.com/mbostock/d3/wiki/Force-Layout* (besucht am 01. 09. 2013).

[Gra] *Die Anfänge der Graphentheorie*. URL: *http://www.mathe.tu-freiberg.de/~hebisch/cafe/graphentheorie.html* (besucht am 04. 09. 2013).

[Hbr] *Handlebars.js Tutorial: Learn Everything About Handlebars.js JavaScript Templating*. 2013. URL: *http://javascriptissexy.com/handlebars-js-tutorial-learn-everything-about-handlebars-js-javascript-templating/* (besucht am 30. 08. 2013).

[O'R05] Tim O'Reilly. *What is Web 2.0? Design Patterns and Business Models for the Next Generation of Software*. 2005. URL: *http://www.oreilly.de/artikel/web20.html* (besucht am 13. 08. 2013).

[Wir] *Website wireframe*. URL: *http://en.wikipedia.org/wiki/Website_wireframe* (besucht am 02. 09. 2013).

[neo13] The Neo4j Team. *The Neo4j Manual v1.9.2*. Version 1.9.2. 2013. URL: *http://docs.neo4j.org/chunked/1.9.2/index.html* (besucht am 12. 08. 2013).

Bildquellen

[Fis] *Fisheye Distortion*. URL: *http://bost.ocks.org/mike/fisheye/* (besucht am 06. 09. 2013).

[Koe] *Königsberger Brückenproblem*. URL: *http://www.matheprisma.uni-wuppertal.de/Module/Koenigsb/Einleitu/Einleitu.htm* (besucht am 04. 09. 2013).

[Sno] *Cholera map*. Lizenz: Public Domain. URL: *http://nl.wikibooks.org/wiki/Bestand:Snow-cholera-map.jpg* (besucht am 02. 09. 2013).

[Tre] *d3.layout.tree*. URL: *http://mbostock.github.io/d3/talk/20111018/tree.html* (besucht am 05. 09. 2013).

A Anhang

A.1 c.cat Projektskizze

Prof. Dr. Stefanie Rathje (HTW Berlin) und
Prof. Dr. Hartmut Schirmacher (Beuth Hochschule Berlin)

1. Inhalte des Konzepts Multikollektivität

Jeder Mensch ist gleichzeitig Teil zahlreicher Kollektive, besitzt also zahlreiche Gruppenzugehörigkeiten (z.B. Deutsche, Frau, Professorin, Chormitglied, Fan einer Rockband, Berlinerin, Anhängerin einer bestimmten Partei, etc.). Diese Gruppenzugehörigkeiten sind teilweise fest, teilweise veränderbar. In modernen Gesellschaften lassen sich Gruppenzugehörigkeiten relativ einfach neu hinzu addieren oder auch wieder aufgeben.

Alle Gruppenzugehörigkeiten üben einen kulturellen Einfluss auf das Individuum aus und beeinflussen zusammen mit seiner Biografie und seinen genetischen Vorbedingungen die individuelle Identität. Individuen können sehr unterschiedliche und auch widersprüchliche Zugehörigkeiten in ihrer Identität vereinen. Je nach Situationszusammenhang wird die eine oder andere Zugehörigkeit "virulent".

2. Bedeutung des Konzepts in Interkulturellen Trainings

Die Vorstellung von Multikollektivität ermöglicht es, die veraltete Vorstellung einer einzigen Kulturzugehörigkeit und damit verbunden, die Vorstellung, man könne einem Individuum bestimmte Eigenschaften aufgrund seiner Kulturzugehörigkeit zuordnen, zu überwinden.

Durch die Darstellung und den Vergleich individueller Zugehörigkeiten in diversen Gruppen können Anknüpfungspunkte zwischen Individuen geschaffen werden („Mehmet hat ja auch zwei Töchter.“, „Han ist auch Grunge-Fan“), so dass die primäre Zuordnung von Menschen zu Nationen oder Ethnien zugunsten einer multiperspektivischen Sichtweise überwunden werden kann.

3. Anforderungen an ein Tool zur Erfassung und Visualisierung von Multikollektivität

- Computergestützte Erfassung und Speicherung einer „Ontologie“ jedes Teilnehmers, die Aufschluss über seine Kollektivzugehörigkeiten gibt.
- Analyse von Gemeinsamkeiten (gemeinsame Gruppenzugehörigkeiten) auf unterschiedlichen Ebenen zwischen den Teilnehmern in multiplen Nähe-Ferne- Relationen
- Visualisierung der multiplen Nähe-Ferne-Relationen der Teilnehmer

4. Potentielle Ansätze der Informatik

Ontologie

Zu Beginn der Konzepterstellung steht der Entwurf einer beispielhaften Ontologie, die die ein „Raster“ für die Einordnung der Gruppenzugehörigkeiten eines Individuums vorgibt. Die Verwendung einer solchen Ontologie schränkt die anfängliche Flexibilität des Ansatzes ein, ermöglicht aber im Sinne des Rapid Prototyping den Entwurf erster Software-Applikationen zur Analyse und Visualisierung gemeinsamer Interessen und Gruppenzugehörigkeiten auf mehreren Auflösungs- und Abstraktionsebenen (entsprechend den verschiedenen Ebenen der Ontologie).

Analyse und Visualisierung

Für die eigentliche Analyse und Visualisierung ergeben sich zahlreiche potentielle Ansätze, von der Sicht des Individuums bis hin zur globalen Betrachtung. Mit Hilfe von weichen Clusteranalyse-Verfahren soll gezielt ausgewertet werden, wo sich Schwerpunkte der Gemeinsamkeiten der Teilnehmer abzeichnen. Diese Cluster sollen dann mittels geeigneter (und zu erforschender) Verfahren der algorithmischen Informationsvisualisierung dargestellt werden; dabei sollen die optimale Informationsaufbereitung und -vermittlung durch die graphische Darstellung sowie die interaktive Exploration einzelner Zusammenhänge im Vordergrund stehen.

Zu Beginn liegt der Schwerpunkt der Arbeit auf der graphischen und interaktiven Darstellung der Analyseergebnisse. Im späteren Verlauf ist es denkbar, die zugrundeliegende Ontologie entweder sehr stark zu verallgemeinern oder durch eine "flache"Menge von Begriffen mit unscharfen Interrelationen zu ersetzen. Die multidimensionalen Beziehungen der unscharfen Schlagwortmenge sollen automatisch durch Einbeziehung von zusätzlichem Umweltwissen erfolgen (z.B. welche Musikrichtungen oder Sportarten ähneln sich mehr als andere).

A.2 Categories of collectivity

What do you identify with considering...

1. your profession
2. your education
3. your family
4. your sexual identity
5. geographical places
6. organizations or groups you are a part of
    - e.g. political parties
    - e.g. clubs
    - e.g. charity organizations
7. the things you do in your freetime
    - e.g. the sports you like to do or watch
    - e.g. the books or magazines you like to read
    - e.g. the media you like to use or watch
    - e.g. the arts or handicrafts you like to do or enjoy
    - e.g. other aspects of your freetime
8. the things you own
9. brands/products you like to buy
10. your world-view and spiritual life