Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik
FH Zentralschweiz

Prototyping einer webbasierten Visualisierung zur intuitiven Interaktion mit einem Wissensnetzwerk

Informatikprojekt

Andreas Waldis, Patrick Siegfried

30. Januar 2017

Projektpartner & Betreuung: Dr. Michael Kaufmann
Experte: Prof. Dr. Bernhard Hämmerli

Departement Informatik, Hochschule Luzern

Eigenständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel verwendet haben. Sämtliche verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Luzern, 30. Januar 2017

Andreas Waldis

Patrick Siegfried

Abstract

Im Rahmen des Informatikprojekts wurde ein Prototyp für die Visualisierung und Interaktion von ***Netzwerken*** erarbeitet. Dazu wurden mithilfe der Spezifikation von funktionalen Anforderungen verschiedene ***Frameworks*** für die Grundlage der Entwicklung evaluiert. Anschliessend wurde die für den Anwendungsfall beste Wahl getroffen.

Der Prototyp bietet eine Benutzeroberfläche, welche sowohl auf kleinen und grossen Bildschirmen, mit Touch-Gesten oder der Maus und der Tastatur, verwendet werden kann. Der resultierende Prototyp wurde als selbständiges Software-Paket entwickelt. Dadurch kann dieser in beliebigen Applikationen verwendet werden. In der letzten Projektphase wurde der Prototyp schlussendlich in den bestehenden ***ikc-core*** integriert.

Inhaltsverzeichnis

1 Einleitung **1**

2 Projektmanagement **3**
2.1 Ausgangslage 3
2.1.1 Internationale Projektorganisation 4
2.1.2 Werkzeuge 4
2.2 Scope 5
2.3 Projektstrukturplan 5
2.4 Rahmenplan 8
2.5 Projektziele 10
2.6 Anforderungen 10
2.7 Risikoanalyse 13
2.8 Lieferobjekte 15
2.9 Stories 16
2.10 Testkonzept 17

3 Lösungsdesign **19**
3.1 User Interface Design 19
3.1.1 Umgang mit dem Netzwerk 19
3.1.2 Komponenten der Benutzeroberfläche 20
3.1.3 Aktionen 22
3.1.4 Integration 24
3.2 Technische Ausgangslage 25
3.3 Architektur 26
3.4 Schnittstellen 28
3.4.1 GraphScreenProps 29
3.4.2 NodeInformationProvider 29
3.4.3 OperationService 30
3.4.4 IdentityService 32

3.4.5 DialogFactory . . . 32
3.4.6 Dialog-Schnittstellen . . . 34
3.4.7 SearchFieldFactory . . . 37
3.5 Model . . . 38
3.6 Framework-Auswahl . . . 43
3.6.1 Kriterien-Katalog . . . 44
3.6.2 Ergebnisse . . . 46
3.6.3 Bewertungen . . . 46

4 Implementation **51**
4.1 Übersicht . . . 51
4.2 Technologie . . . 51
4.2.1 Typescript . . . 51
4.2.2 React . . . 52
4.2.3 Datenfluss . . . 55
4.2.4 Material Design . . . 56
4.2.5 Webpack . . . 56
4.3 Interkation . . . 57
4.4 Drag'n'Drop . . . 60
4.5 Visualisierung . . . 65
4.5.1 Manipulation des Netzwerks . . . 66
4.5.2 Kontextmenüs . . . 70
4.6 Integration . . . 73

5 Schlussfolgerungen **77**
5.1 Erkenntnisse . . . 77
5.2 Lessons learned . . . 78
5.3 Ausblick . . . 79

Literaturverzeichnis **81**

Abbildungsverzeichnis **86**

A Appendix **91**
A.1 Testfälle . . . 91
A.2 User-Stories . . . 93
A.3 Wichtigste Protokolle . . . 97
A.4 Arbeitsjournal . . . 103
A.5 Aufgabenstellung . . . 108

Kapitel 1

Einleitung

Im Forschungsprojekt ***IKC***[1] wird ein Prototyp für den Umgang mit einem plattformübergreifenden Wissensnetzwerk entwickelt. Obwohl die grundlegende Datenbasis ebenfalls in Form eines Netzwerks aufgebaut ist, wählt die Benutzeroberfläche einen anderen Ansatz: Bisher handelt es sich lediglich um eine technische Konsole. Diese macht für den Benutzer zwar alle existierenden Funktionalitäten zugänglich, jedoch ist sie weder sonderlich effizient noch benutzerfreundlich. Für einen ersten Machbarkeitsnachweis war dies ausreichend. Um den Prototypen aber einem grösseren Nutzerkreis verfügbar zu machen, gibt es Optimierungsbedarf. Auch macht es aus Sicht des Projektteams Sinn, dass die Netzwerkstruktur auch für den Benutzer sichtbar ist. Das Ziel dieser Arbeit ist darum ein Prototyping einer webbasierten Visualisierung zur intuitiven Interaktion mit einem Wissensnetzwerk.

Zum besseren Verständnis werden potentiell unbekannte oder projektspezifische Begriffe kursiv und fett dargestellt. Zu jedem Begriff ist eine kurze Beschreibung im Glossar (Kapitel 5.3) am Ende der Dokumentation zu finden. Im Anhang sind neben der Aufgabenstellung (Abschnitt A.5) auch die wichtigsten Sitzungsprotokolle (Abschnitt A.3) und das Arbeitsjounrnal (Abschnitt A.4) beigelegt. Zusätzlich zu diesem Dokument werden noch die beiden Handbücher für die *Entwicklung* und die *Handhabung* mitgeliefert.

[1] ***Intuitive Knowledge Connectivity***

Kapitel 2

Projektmanagement

Das folgende Kapitel beschäftigt sich mit der Führung und Kontrolle des Projekts. Das Projektmanagement, wie auch die spätere Entwicklung funktionieren agil. Um möglichst effizient auf Änderungen reagieren zu können, werden entsprechende Hilfsmittel verwendet.

2.1 Ausgangslage

Aus dem Forschungsprojekt ***Intuitive Knowledge Connectivity***[1] ging neben einer Publikation[2] auch ein Software-Prototyp[3] (***ikc-core***) hervor. Das Projekt beschäftigt sich mit einem intuitiven Umgang mit Daten aus diversen Cloud-Dienstleistern, beispielsweise ***Dropbox***, ***Evernote*** und vielen mehr. Diese werden, ebenfalls ***cloudbasiert***, gespeichert und verteilt. Der Mehrwert entsteht durch die Verknüpfung der verschiedenen Daten zu einem gesamtheitlichen Ganzen. Diese Verknüpfungen sind benutzerbasiert und werden mithilfe einer Graph-Datenbank gehandhabt.

Da beide Projektmitarbeiter ebenfalls am Forschungsprojekt tätig sind, ist das Grundlagewissen bereits grösstenteils vorhanden. Die Arbeit am Forschungsprojekt und am ***PAWI*** verlaufen parallel weiter. Um eine adäquate Trennung der beiden Projekte zu gewährleisten, wird im Abschnitt 2.2 näher auf die Gegebenheiten eingegangen.

[1] https://www.hslu.ch/en/lucerne-university-of-applied-sciences-and-arts/research/projects/detail/?pid=3334

[2] (Kaufmann et al., 2016)

[3] http://demo.ikc.today/nodeDetail.html

2.1.1 Internationale Projektorganisation

Damit Andreas Waldis ein Team-Mitglied während des Projekts in den Vereinigten Staaten im Austauschsemester weilt, müssen einige zusätzliche organisatorische Massnahmen ergriffen werden. Diese umfassen die folgenden Tools und Arbeitsweisen:

- Die Zeitverschiebung zwischen Luzern (*UTC+01:00*) und West Lafayette (*UTC-05:00*) beträgt sechs Stunden. Daher werden Sitzungen überwiegend nachmittags nach 14:00 angesetzt.
- Projektplanung: Die Projektdauer wurde für diesen speziellen Fall um sechs Wochen auf Ende Januar erweitert. Dies ermöglicht die verschiedenen Teammitglieder entsprechend ihrer Auslastung zu arbeiten. Insbesondere für Andreas Waldis, welcher bereits im Dezember die Abschlussprüfungen hat.

2.1.2 Werkzeuge

Es werden verschiedene Werkzeuge verwendet, um eine möglichst optimale Kommunikation und Kollaboration zu gewährleisten. Dazu gehören:

1. **Skype**[4] & **Discord**[5]: Beides Werkzeuge für Video- und Audio-Sitzungen. Skype ist wohlbekannt. Discord hingegegen ist spezialisiert für Gamer und beinhaltet ein umfangreicheres Chat-System.
2. **Targetprocess**[6]: Vergleichbar mit dem bekannten ScrumDo[7]. Im Gegensatz dazu verfügt Targetprocess jedoch über eine moderne Benutzeroberfläche und scheint für den vorliegenden Verwendungszweck allgemein geeigneter zu sein.
3. **Sharelatex**[8]: Online Kollaborationstool für Latex. Anstelle von lokalen Installationen in einem gemeinsamen Versionsverwaltungssystem, wird einfach direkt online am selben Dokument gearbeitet.
4. **draw.io**[9]: Werkzeug, um online Diagramme zu zeichnen. Diese können ebenfalls gemeinsam und gleichzeitig gezeichnet wer-

[4] https://www.skype.com/de/
[5] https://discordapp.com/
[6] https://www.targetprocess.com
[7] http://www.scrumdo.com
[8] https://www.sharelatex.com
[9] https://www.draw.io

den. Zusätzlich ist es ideal, um schnell und einfach alles Mögliche zu zeichnen.

2.2 Scope

Die Schwierigkeit liegt zu einem grossen Teil in der Abgrenzung zum parallellaufenden Projekt ***IKC***. Grundsätzlich gehört jegliche Arbeit im zweidimensionalen Bereich zum ***PAWI***. Der bestehende ***ikc-core*** wird soweit vorbereitet, dass die Visualisierung ohne Zusatzaufwand integriert werden kann. Hierzu gehören Arbeiten betreffend Schnittstellen, Absenden von allfälligen ***Event***s und dergleichen. Das Ziel ist die Entwicklung eines Zusatzmoduls zum ***ikc-core***, welches mittels definierten Schnittstellen ein- und angebunden werden kann. Die genaue Spezifikation ist im Lösungskonzept noch zu erarbeiten.

Weiter ist zu betonen, dass die intuitive Bedienung (***Usability*** oder auch ***User Experience***) nach bestem Wissen und Gewissen angestrebt wird. Allerdings handelt es sich bei den Projektmitarbeitern um Informatiker ohne grundlegende Kenntnisse in den angesprochenen Bereichen. Die Benutzeroberfläche wird darum nach dem subjektiven Befinden des Projektpartners und den beiden Projektmitarbeitern entwickelt.

2.3 Projektstrukturplan

Die Abbildung 2.1 gewährt einen Überblick über das Projekt. Sie stellt die wichtigsten Bereiche und Phasen dar, in welche die Arbeit grob eingegliedert werden kann:

1. Die **Projektführung** beinhaltet die Planung des Vorhabens über den gegebenen Zeitraum. Ständige Kontrolle des Ist- gegenüber dem Soll-Zustand kann gegebenenfalls zur Steuerung oder Anpassungen des Zeitplans führen. Im Gegensatz zu den anderen Bereichen wird die Projektführung über die volle Projektdauer ausgeführt. Da das Projekt agil organisiert ist, liegt das Augenmerk auf der Priorisierung der Anforderungen.

2. In der **Konzeption** werden neben den Anforderungen auch mögliche Lösungsansätze in den Bereichen Architektur, Schnittstellen und ***User Interface/User Experience*** gesammelt.

3. Nach der Recherche von Lösungsansätzen muss allenfalls die Auswahl an Möglichkeiten eingeschränkt werden. Eine ansch-

liessende Testphase erleichtert die **Evaluation und Auswahl** der Lösungsvariante.

4. Nun beginnt die Phase der eigentlichen **Entwicklung**. Auf Basis der bestehenden Tests wird die Visualisierung mit allen erforderlichen Komponenten erarbeitet. Der Schwerpunkt liegt auf der Gestik und dem ***Responsive Design***. Ein abschliessendes Testing des Systems stellt sicher, dass die Integration beginnen kann.
5. Sobald die Visualisierung reibungslos funktioniert, wird sie in den bestehenden ***ikc-core*** **integriert**. Alle Anforderungen werden erfüllt und anschliessend getestet.
6. Nachdem die Entwicklungsarbeiten abgeschlossen sind, folgt der **Projektabschluss**. Dabei wird die endgültige Version des Projektreports erstellt und die Abschlusspräsentation gehalten.

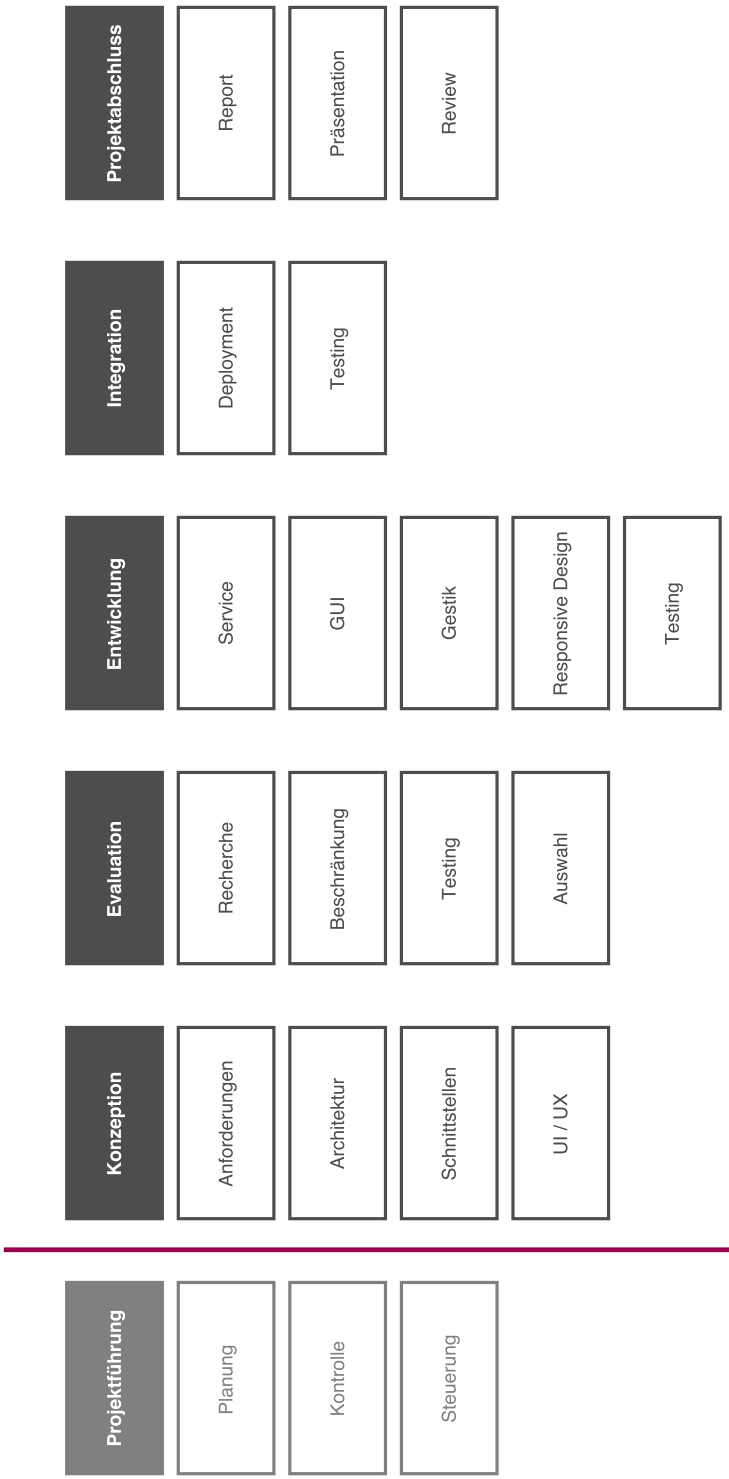


Abbildung 2.1: Projektstrukturplan

2.4 Rahmenplan

Die Rahmenplanung, basierend auf dem Projektstrukturplan (Abbildung 2.2), repräsentiert die zeitliche Planung des Projekts. Dabei werden Kalenderwochen anstelle von Daten oder Schulwochen verwendet. Dies aufgrund des internationalen Rahmens, der damit verbundenen Zeitverschiebung und unterschiedlichen Stundenplänen. Enthalten sind alle Projektphasen, Sprints und Meilensteine, als auch alle Lieferobjekte welche im Abschnitt 2.8 weiter ausgeführt werden. Die Dauer der Sprints wird bewusst unterschiedlich ausgestaltet, um den verschiedenen Projektphasen und deren Inhalten Rechnung zu tragen.

Eine grosse Rolle in der Rahmenplanung spielen die Meilensteine. Sie unterteilen das Projekt in Phasen, welche dadurch klar voneinander getrennt sind. Ebenfalls sind sie eine wichtige Orientierungshilfe im Projekt und weisen den Weg damit das Projekt erfolgreich abgeschlossen werden kann. Die Tabelle 2.1 listet die Meilensteine auf.

ID	Datum	Beschreibung
M1	19.09.2016	Administrativer Meilenstein: Kickoff
M2	09.10.2016	Administrativer Meilenstein: Projektplanung abgeschlossen
M3	16.10.2016	Entwicklung Meilenstein: Schnittstellen definiert
M4	06.11.2016	Entwicklung Meilenstein: Evaluation Entscheid
M5	25.12.2016	Entwicklung Meilenstein: Funktionsfähige Oberfläche umgesetzt
M6	15.01.2017	Entwicklung Meilenstein: Integration in ***ikc-core*** abgeschlossen
M7	22.01.2017	Administrativer Meilenstein: 95% erreicht
M8	30.01.2017	Administrativer Meilenstein: PAWI Bericht Abgabe
M9	31.01.2017	Administrativer Meilenstein: Präsentation

Tabelle 2.1: Meilensteine

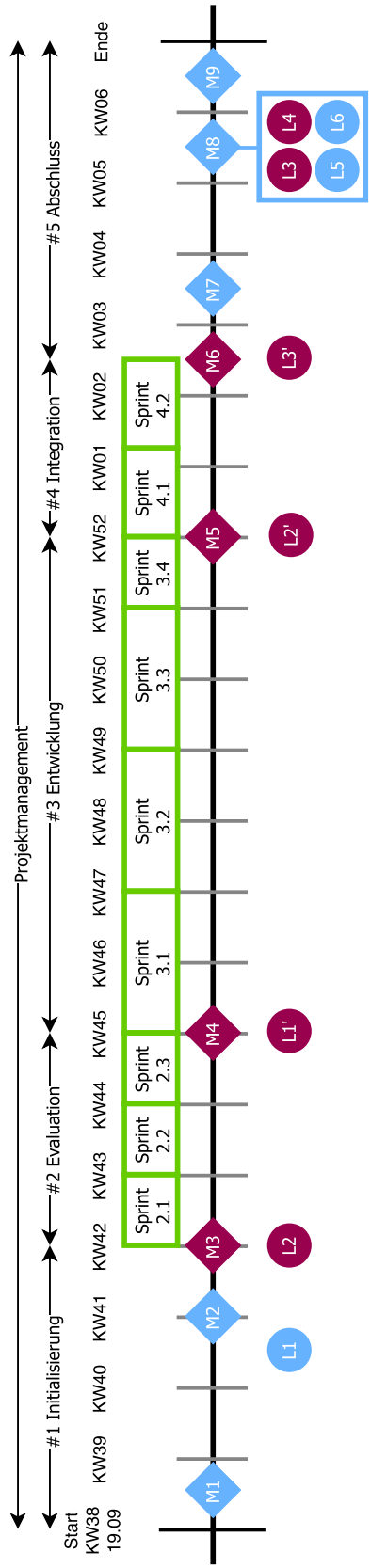


Abbildung 2.2: Rahmenplan

2.5 Projektziele

Projektziele werden definiert, um den Erfolg an einigen ausgewählten Punkten zu überprüfen und sicherstellen zu können. Sie wurden in Absprache mit dem Kunden definiert. Die Ziele sind in der folgenden Tabelle 2.2 aufgelistet.

ID	Beschreibung
Z1	Ergänzung zur bestehenden Benutzeroberfläche.
Z2	Entwicklung nach dem Konzept ***mobile first***.
Z3	Intuitive, visuelle und effiziente Interaktion mit einem ***Netzwerk***.
Z4	Übersichtliche Visualisierung der ***Nodes*** und ***Links***.
Z5	Erweiterung der Funktionalität mittels Sichten (***View***s).

Tabelle 2.2: Projektziele

2.6 Anforderungen

Für die weitere Unterteilung in Arbeitspakete und *Stories* werden die Anforderungen zunächst in Prosa gesammelt. Diese entstammen dem Kundenworkshop und der Aufgabenstellung, sind im Sinn der Projektziele (Abschnitt 2.5). Die Anforderungen werden unterschieden in funktionale und nicht-funktionale Anforderungen. Die funktionalen Anforderungen definieren direkt die Eigenschaften, (Tabelle 2.4). Im Gegensatz dazu definieren nicht-funktionale Anforderungen die Leistung und die Randbedingungen, aufgelistet in Tabelle 2.5.

Die Priorisierung erfolgt nach dem ***MoSCoW-System***:

#	Priorität	Beschreibung
M	Must Have	Bei dieser Anforderung handelt es sich um ein Muss, höchste Priorität.
S	Should Have	Diese Anforderung wird erwartet, normale Priorität.
C	Could Have	Tiefste Priorität, desiderata.

Tabelle 2.3: MosCow-Priorisierung

ID	Priorität	Beschreibung

A1.1	M	Die zugrundeliegende Datenbasis kann mittels eines ***Netzwerks*** visualisiert werden. Jenes besteht aus Knoten und Kanten, welche gerichtet und beschriftet sind.
A1.2	M	Die zu erarbeitende Visualisierung kann bidirektional mit dem ***ikc-core*** interagieren. Änderungen am ***ikc-core*** sind in der Visualisierung sichtbar. Es ist jedoch auch möglich Änderungen direkt in der Visualisierung vorzunehmen.
A1.3	S	Mittels der Visualisierung können die grundlegenden Datenbankoperationen *CREATE*, *READ*, *UPDATE* und *DELETE* (CRUD) teilweise direkt auf der Datenbasis des ***ikc-cores*** angewendet werden.
A1.3.1	S	Es können sowohl Knoten aus der Visualisierung als auch aus der Datenbasis[10] gelöscht werden. Diese Vorgänge können klar unterschieden werden.
A1.3.2	M	Knoten ohne Kanten können in der Visualisierung abgebildet werden.
A1.3.3	M	Knoten können in der Visualisierung erstellt werden.
A1.4	M	Der Prototyp kann auf Smartphones und Tablets benutzt werden.
A1.5	S	Der Prototyp kann auf Laptops und Desktop-Computern benutzt werden.
A1.6	M	Mit Hilfe verschiedener Kriterien (beispielsweise Kontext oder Nachbarschaft) kann ein Teil eines ***Netzwerks*** visualisiert werden.[11]
A1.6.1	S	Diese Visualisierungen (ein Teil des ***Netzwerks***) können unabhängig von der Datenbasis persistiert werden. Es können, zusätzlich zu den Knoten und Kanten, auch deren relative Positionen in der Visualisierung gespeichert werden.
A1.7	C	Der Prototyp kann (unter anderem) mit ***Drag'n'Drop*** bedient werden.

[10] Hier ist die Datenbasis des ***ikc-cores*** gemeint.

[11] Die Visualisierung eines Teils eines ***Netzwerks*** wird später als ***View*** bezeichnet.

A1.7.1	S	Bidirektionale und unbeschriftete Verknüpfungen zwischen visualisierten Knoten können erstellt werden.
A1.7.2	M	Ein, in der bestehenden Benutzeroberfläche[12], mittels der Suche gefundener Knoten kann direkt in die Visualisierung übernommen werden.
A1.7.3	S	Ein mittels der Suchfunktion gefundener Knoten kann auf einen bestehenden Knoten gezogen werden, um mit diesem eine Kante zu bilden.
A1.7.4	S	Knoten können innerhalb des Diagramms frei positioniert werden.
A1.8	M	Die Visualisierung kann innerhalb der bestehenden ***ikc-core*** Oberfläche genutzt werden.
A1.9	S	In der Visualisierung könnnen auch Knoten mit mehr als sieben Kindknoten dargestellt werden: Sind mehr als sieben Kindknoten vorhanden, wird anstelle einer Repräsentation jedes einzelnen Knotens auf eine Liste gewechselt. Diese Liste kann zusätzlich mit einer Such- und oder Filterfunktion versehen werden.
A1.10	C	Die von einem Knoten aus- und eingehenden Kanten können auf- und zugeklappt (sichtbar-unsichtbar) werden.
A1.10.1	C	Einzelne Kanten können individuell zugeklappt/ausgeblendet werden.

Tabelle 2.4: Funktionale Anforderungen

ID	Priorität	Beschreibung
A2.1	M	Bestehende Komponenten des ***ikc-cores*** werden, wo möglich, wiederverwendet bzw. erweitert (nachhaltige Entwicklung).
A2.2	M	Die Visualisierung kann in verschiedenen Projekten wiederverwendet werden.
A2.3	M	Die Visualisierung wird im ***ikc-core*** integriert (Deployment).

[12] ***ikc-core***

A2.4	M	Die Visualisierung wird parallel zum ***ikc-core*** weiterentwickelt. Die Basis für die Visualisierung bildet ein Klon der bestehenden Codebasis aus dem Versionkontrollsystem. Ist die Entwicklung abgeschlossen, werden die beiden Entwicklungsstände mittels eines neuen Astes (Branch) zusammengeführt.
A2.5	M	Jegliche Daten werden auf der Dropbox des Benutzers persistiert.
A2.6	S	Es wird eine intuitive, effiziente Bedienung angestrebt. Ein Mass für die Effizienz: $\frac{\text{Taps}}{\text{Task}}$
A2.7	M	Randbedingungen 180h pro Person
A2.8	S	Die Arbeit können in einem Arbeitsjournal, mindestens mit Angaben von Datum, Anzahl Stunden, Arbeitsschritt/Thema, nachverfolgt werden.

Tabelle 2.5: Nicht funktionale Anforderungen

2.7 Risikoanalyse

In folgender Tabelle 2.6 werden mögliche Risiken behandelt. Die Wahrscheinlichkeit ist mit P abgekürzt. R steht für Risiko und S für den Schaden, welcher mittels $P * R = S$ berechnet wird.

ID	Beschreibung	P	R	S
R1	Wie in Unterabschnitt 2.1.1 bereits angesprochen wird das Projekt in einem **internationalen Rahmen** durchgeführt. Dies birgt einige Herausforderungen und bringt auch Riksiken mit sich: Die grösste Schwierigkeit bildet sicherlich die Zeitverschiebung. Aber auch die geographische Distanz und die lediglich im digitalen Rahmen gehaltenen Sitzungen sind nicht zu unterschätzen. Um dieser Problematik zu entgegnen, werden diverse Kommunikationswege, vorwiegend verschiedenen Online-Plattformen, benutzt. Diese sollen trotz den Gegebenheiten die Kollaboration und den Austausch unterstützen und fördern.	1	3	3
R2	Die Abgrenzung vom laufenden Projekt (vgl. Abschnitt 2.2) ist klar festzulegen und einzuhalten. So können Überschneidungen und Unklarheiten verhindert werden. Vollständige und detaillierte Arbeitsjournale, wie auch Protokolle bieten dabei eine wichtige Hilfestellung.	2	1	2
R3	Die Zahl der Anforderungen ist hoch. Im Rahmen der **360 zu leistenden Stunden** ist es wichtig, sich nicht in Details oder in der Ausarbeitung jeglicher Finessen zu verlieren. Darum ist es umso wichtiger, die Anforderungen klar zu priorisieren und auch entsprechend abzuarbeiten.	3	1	3

Tabelle 2.6: Risikoanalyse

2.8 Lieferobjekte

Neben den in der Aufgabenstellung vorgegebenen Lieferobjekte (Tabelle 2.7) sind noch zusätzliche, interne Lieferobjekte (Tabelle 2.8) festlegt. Diese sind lediglich als Unterstützung der Projektkontrolle, eine Art Orientierungshilfe, gedacht.

ID	Datum	Beschreibung
L1	30.09.2016	Projektplanung.
L2	07.10.2016	Anforderungskatalog.
L3	17.10.2016	Lösungskonzept inkl. Schnittstellendefinition.
L4	30.01.2017	Funktionsfähige Software mit folgenden Eigenschaften: • Integriert bestehender Prototyp auf Basis *React* mit allen ***CRUD***[13] Funktionen und Schnittstellen zu ***Dropbox*** und ***Evernote***. • Möglichkeit zur visuellen Interaktion mit einem Teil eines ***Netzwerks***: Knoten können per ***Drag'n'Drop*** verknüpft werden • Die Applikation läuft auf dem Mobile, Tablet, Laptop und Desktop (in Priorität der Reihenfolge) → responsive CSS Template[14] verwenden.
L5	30.01.2017	Dokumentierter Sourcecode (für Methoden und Parameter).
L6	30.01.2017	Benutzerhandbuch.
L7	30.01.2017	***PAWI***-Bericht.

Tabelle 2.7: Vorgegebene Lieferobjekte

ID	Datum	Beschreibung
L1'	07.11.2016	Abgeschlossene Evaluation der verschiedenen ***Frameworks*** zur Umsetzung der Visualisierung.

[13] Create Read Update Delete
[14] ***Responsive Design***

L2′	26.12.2016	Funktionsfähige Visualisierung gemäss den funktionalen Anforderungen A1.3, A1.4, A1.5, A1.6, A1.7, A1.9 und A1.10. Bereit für die Integration in den ***ikc-core***.
L3′	13.01.2017	Abgeschlossene Integration der Visualisierung in den ***ikc-core*** und Erfüllung aller Anforderungen insbesondere A1.1, A1.2.

Tabelle 2.8: Zusätzliche, interne Lieferobjekte

2.9 Stories

Die User-Stories repräsentieren alle Arbeitspakete, welche über die gesamte Projektdauer geplant wurden. Diese werden nicht nur im klassischen Sinne für die Klassifizierung von Applikationsfunktionen, sondern auch für konzeptionelle Aufgaben verwendet. Insgesamt haben wurde der Aufwand mit **256** Punkte beziffert, wobei ein Punkt circa einer Stunde entspricht. Dies entspricht auch etwa dem resultierenden Aufwand von **276** Punkten. Der Mehraufwand konnte dank einiger Reserven gut kompensieren werden. Dieser entstand vor allem in der Umsetzung der ***Drag'n'Drop*** Gesten und dem Datenaustausch zwischen ***ikc-core*** und Visualisierung. Alle User-Stories sind in der Tabelle 2.9 detailliert aufgelistet und die weiterführenden Beschreibungen sind in der Tabelle A.2 zu finden.

ID	Name	Geplant	Effektiv	Phase	Sprint
S1	Defintion Schnittstellen	10 pt	12 pt	Evaluation	2.1
S2	Grobauswahl	20 pt	17 pt	Evaluation	2.1
S3	Detail Beurteilung zwei ***Frameworks***	4 pt	5 pt	Evaluation	2.3
S4	Detail Beurteilung zwei ***Frameworks***	4 pt	4 pt	Evaluation	2.2
S5	Definition Standard Szenario	4 pt	3 pt	Evaluation	2.1
S6	Mook Up erstellen	8 pt	10 pt	Evaluation	2.3
S7	Basic Setup	3 pt	4 pt	Entwicklung	3.1
S8	***Drag'n'Drop***	20 pt	29 pt	Entwicklung	3.4
S9	PositionUpdate	4 pt	4 pt	Entwicklung	3.1
S10	NewLink	4 pt	4 pt	Entwicklung	3.2

S11	Core Context Menu	15 pt	22 pt	Entwicklung	3.1
S12	Node Context Menu	25 pt	23 pt	Entwicklung	3.2
S13	LinkCollapse	10 pt	11 pt	Entwicklung	3.3
S14	Show/Hide Labels	5 pt	5 pt	Entwicklung	3.4
S15	Integration IKC	45 pt	38 pt	Integration	4.1
S16	Datenaustausch	20 pt	26 pt	Integration	4.2
S17	Persistenz	20 pt	23 pt	Integration	4.2
S18	Dialoge	20 pt	22 pt	Integration	4.1
S19	SearchFields	15 pt	14 pt	Integration	4.1
	Total	**256 pt**	**276 pt**		

Tabelle 2.9: User Stories

2.10 Testkonzept

Basierend auf den Anforderungen wurden die verschiedenen Testfälle definiert. Diese sind hier zusammengefasst. Konkret handelt es sich um die folgenden Testfälle (Tabelle 2.10), welche im Abschnitt A.1 genauer beschrieben sind.

ID	Kurzbeschrieb
T1	Neuer ***Node*** erstellen und darstellen.
T2	Neuer ***Node*** erstellen, darstellen und mit einem bereits dargestellten ***Node*** verbinden.
T3	Bestehender ***Node*** in der Visualisierung darstellen.
T4	Bestehender ***Node*** mit einem bereits dargestellten ***Node*** verbinden.
T5	Bestehender ***Node*** in der Visualisierung darstellen (***Drag'n'Drop***).
T6	Bestehender ***Node*** mit einem bereits dargestellten ***Node*** verbinden (***Drag'n'Drop***).

T7	***Node*** ausblenden.
T8	Alle ausgehenden ***Links*** ausblenden.
T9	Alle ausgehenden ***Links*** einblenden.
T10	Einzelner ausgehender ***Link*** einblenden.
T11	Verschiedene ausgewählte ***Links*** ausblenden.
T12	Neue ***View*** erstellen.
T13	Existierende ***View*** öffnen.
T14	***Node*** Informationen in der Datenbasis aktualisieren.
T15	***Node*** in der Datenbasis löschen.
T16	***Link*** in der Datenbasis löschen.
T16	***Link*** in der Datenbasis erstellen.
T17	***Node*** mit mehr als sieben ausgehende ***Links*** darstellen.
T18	Label der ***Links*** aus- und einblenden.

Tabelle 2.10: Testfälle

Kapitel 3

Lösungsdesign

Das Lösungsdesign beinhaltet die Grundlagen für die erfolgreiche Umsetzung des Prototyps. Vor der eigentlichen Implementation wird das Vorhaben auf konzeptioneller Ebene genauer betrachtet. Die Definition der Schnittstellen, besonders die Verminderung der Kopplung zwischen der Visualisierung und dem ***ikc-core***, ist hier ein wichtiger Punkt.

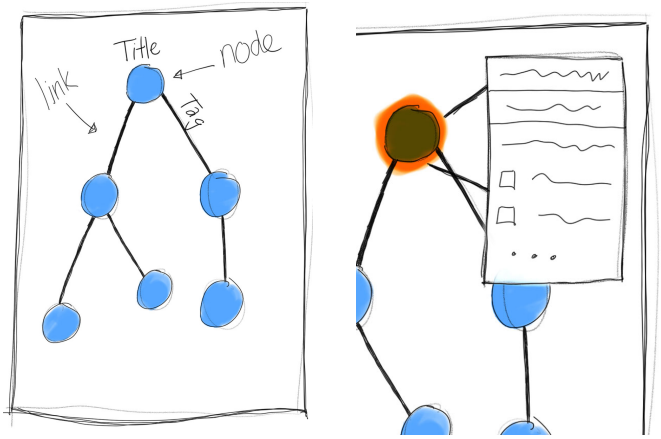
3.1 User Interface Design

Nachfolgend werden wichtige Punkte zum Umgang mit dem ***Netzwerk*** und zur Integration der Visualisierung aufzeigt und weitergehende Überlegungen dazu dargelegt.

3.1.1 Umgang mit dem Netzwerk

Die Benutzeroberfläche orientiert sich hauptsächlich am ***Netzwerk***, bestehend aus ***Nodes*** und ***Links***. Diese besitzen eine Beschriftung in Form des Titels oder in Form des entsprechenden ***Tags*** (Abbildung 3.1a). Die Beschriftung kann zugunsten der Übersichtlichkeit gegebenenfalls ausgeblendet werden.

Wo immer möglich soll die Interaktion nicht über eine entfernte Schaltfläche, sondern direkt am jeweiligen Element in der Visualisierung geschehen. Ein mögliches Beispiel hierfür zeigt Abbildung 3.1b. Am Knoten, wo eine Aktion vorgenommen werden soll, kann mittels eines Klicks (oder Ähnlichem) ein Kontextmenü geöffnet werden. Dieses ermöglicht der Situation entsprechende, weiterführende Funktionalitäten. Entsprechende Möglichkeiten sind auch beim Umgang mit Kanten vorstellbar.



(a) Nodes und Links (b) Node-Menü

Abbildung 3.1: Skizzen: Benutzeroberfläche

3.1.2 Komponenten der Benutzeroberfläche

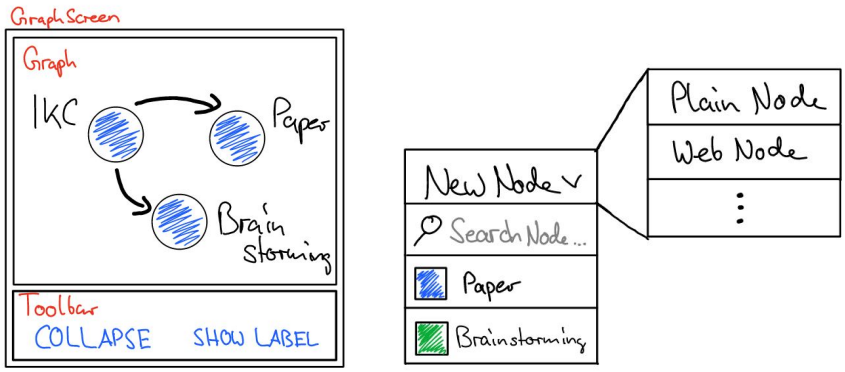
Die Visualisierung verwendet Komponenten, welche Funktionen für den Benutzer zusammenfassen oder als Container für weitere Elemente dienen können. In der späteren Implementation werden diese komplett oder teilweise direkt mit einer *React*-Komponente realisiert.

GraphScreen

Der *GraphScreen* (Abbildung 3.2a) beinhaltet alle wesentlichen Elemente der Visualisierung. Dazu gehört in erster Linie das ***Netzwerk***, in welchem die ***Nodes*** und ***Links*** dargestellt werden. Weiter ist eine Navigationsleiste (*Toolbar*) enthalten, in welcher wichtige Funktionen zur Verfügung gestellt werden.

CoreContextMenu

Das über einen *Rechtsklick* (Desktop) bzw. *Tap* (Mobile) zu öffnende *CoreContextMenu* macht für den gegebenen Kontext weitere Funktionen abrufbar (Abbildung 3.2b). Dabei geht es in erster Linie darum in der Visualisierung neue ***Nodes*** darzustellen. Einerseits können neue ***Nodes*** erstellt, andererseits über die Suchfunktion (auch in der Datenbasis bestehende) gefunden und verwendet werden. Der neu hinzugefügtn Knotenbefindet sich automatisch an jener Position, wo das Menü geöffnet wurde.



(a) GraphScreen (b) CoreContextMenu

Abbildung 3.2: Skizzen: Benutzeroberfläche

NodeContextMenu

Aktionen, welche auf einen spezifischen ***Node*** angewendet werden, sind im *NodeContextMenu* zusammengefasst (Abbildung 3.3). Dazu zählen Aktionen, mit welchen der entsprechende ***Node*** oder dessen ***Links*** bearbeitet werden können. Auch dieses Menü öffnet sich durch einen *Rechtsklick* (Desktop) oder *Tap* (Mobile).

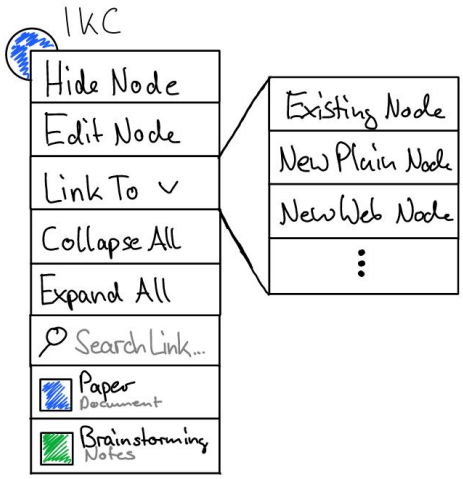


Abbildung 3.3: Skizze: NodeContextMenu

3.1.3 Aktionen

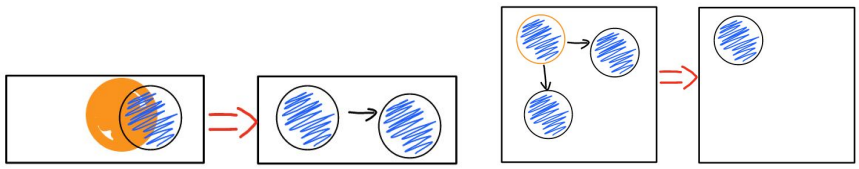
In einem Menu, der Toolbar oder durch ein ***Drag'n'Drop*** können verschiedene Funktionen auf einzelne ***Nodes***, ***Links*** oder das gesamte ***Netzwerk*** angewendet werden. Dazu gehören:

Aktion	Beschreibung	CoreContextMenu	NodeContextMenu	Toolbar	Drag'n'Drop	Click	Tab
Edit Node	Einen ***Node*** bearbeiten.		x			x	
Add Node	Einen existierenden ***Node*** aus der Datenbasis der Visualisierung hinzufügen. Dabei werden alle ***Links*** und deren Ziel-Nodes ebenfalls dargestellt.	x			x		
Hide Node	Einen ***Node*** mit seinen ***Links*** aus der Sicht entfernen (nicht aber aus der Datenbasis).		x				
Delete Node	Das Löschen eines ***Nodes*** aus der Sicht und der Datenbasis. Dies ist nur möglich durch die Bearbeitung des ***Nodes*** in der Detailansicht.						
Delete Link	Das Löschen eines ***Links*** aus der Sicht und der Datenbasis. Dies ist nur möglich durch die Bearbeitung des ***Nodes*** in der Detailansicht.						
New Node	Einen neuen ***Node*** erstellen.	x					

New Link	Einen neuen ***Link*** zwischen zwei ***Nodes*** erstellen. Hierzu wird ein ***Node*** über einen anderen gezogen und losgelassen. (Abbildung 3.4a).				x		
Link To New Node	Neuen ***Node*** erstellen und anschliessend mit einem anderen ***Node*** durch einen ***Link*** verbinden.		x				
Link To Existing Node	***Link*** zu einem existierenden ***Node*** in der Datenbasis erstellen.		x		x		
Collapse All	Alle ausgehenden ***Links*** des entsprechenden ***Nodes*** werden zusammen mit ihren Ziel-Nodes ausgeblendet. Falls der Ziel-Node noch über andere ***Links*** verfügt, wird nur der ***Link*** zwischen dem ***Node*** und dem Ziel-Node ausgeblendet (Abbildung 3.4b).		x				
Select Link	***Links*** auswählen, um eine weitere Aktion auf alle anwenden zu können.					x	x
Collapse Links	Ausgewählte ***Links*** werden zusammen mit ihren Ziel-Nodes ausgeblendet. Falls der Ziel-Node noch über andere ***Links*** verfügt, wird nur der ***Link*** zwischen dem ***Node*** und dem Ziel-Node ausgeblendet (Abbildung 3.5a).			x			

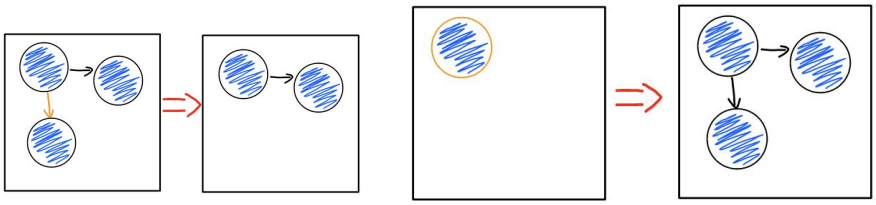
Expand All	Alle ausgehenden ***Links*** eines entsprechenden ***Nodes*** werden zusammen mit deren Ziel-Nodes eingeblendet (Abbildung 3.5b).		x				
Expand Link	Einen ausgehenden ***Link*** eines ***Node*** darstellen.		x				
Show/Hide Labels	Beschreibungen der ***Links*** darstellen oder ausblenden.			x			
Update Position	Ein ***Node*** kann neu positioniert werden.				x		

Tabelle 3.1: Funktionen



(a) New Link (b) Collapse All

Abbildung 3.4: Skizzen: Aktionen



(a) Collapse Link (b) Expand All

Abbildung 3.5: Skizzen: Aktionen

3.1.4 Integration

Aus den Anforderungen geht die Interaktion mit dem ***ikc-core*** als weiterer wichtiger Punkt hervor. Von besonderer Bedeutung ist hier die Bedienung mittels ***Drag'n'Drop***. Wie in Abbildung 3.6 sichtbar, muss

dabei die Grenze zwischen ***ikc-core*** und der Visualisierung (grüne Linie) überwunden werden. Mittels ***Drag'n'Drop*** kann also beispielweise ein ***Node*** aus der ***ikc-core***-Komponente (orange) in die Visualisierung positioniert werden. Dort kann dieser weiter mit dem bestehenden ***Netzwerk*** verknüpft oder als eigenständiger neuer Knoten dargestellt werden.

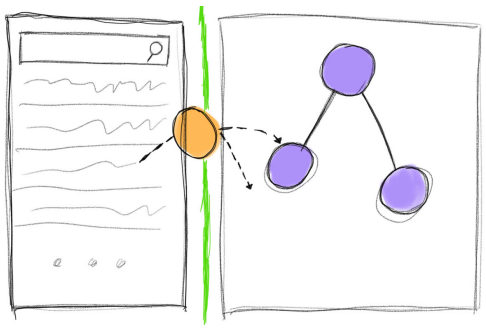


Abbildung 3.6: Grenze ***ikc-core*** - visual

Wo immer möglich, sollen bestehende Komponenten wiederverwendet werden. Dies bietet sich nicht nur bei Abbildung 3.1b in Form eines Kontextmenüs, sondern auch für die Suchfunktion und die Detailansicht an. Für die Desktop-Ansicht wird die Visualisierung mittig in einer dreispaltigen Anordnung in den *ikc-core* eingebettet (Abbildung 3.7). Auf der linken Seite befindet sich die Suche. Auf der rechten Seite die Detailansicht für den in der Visualisierung ausgewählten ***Node***. Dort sind auch weitere Optionen in Form von Menüpunkten verfügbar, beispielsweise sind die ausgehenden ***Links*** erreichbar. Zusätzlich gibt es eine übergeordnete Navigationsleiste, wo wichtige Funktion direkt abrufbar sind.

Bei der Version für Tablets und Smartphones sind, je nach verfügbarer Breite, nur eine oder zwei Spalten sichtbar.

3.2 Technische Ausgangslage

Der ***ikc-core*** funktioniert momentan mit einer rudimentären Benutzeroberfläche. Die Applikationslogik und der Umgang mit der Datenbasis existieren somit bereits. Wie auf dem Komponentendiagramm (Abbildung 3.8) ersichtlich, nutzt der ***ikc-core*** die Visualisierung (*graph-visualization*). Jedoch müssen dazu verschiedene Schnittstellen der Visualisierung implementiert werden. Diese werden von der Visualisierung genutzt, um Operationen an den ***ikc-core*** zu delegieren. Dies

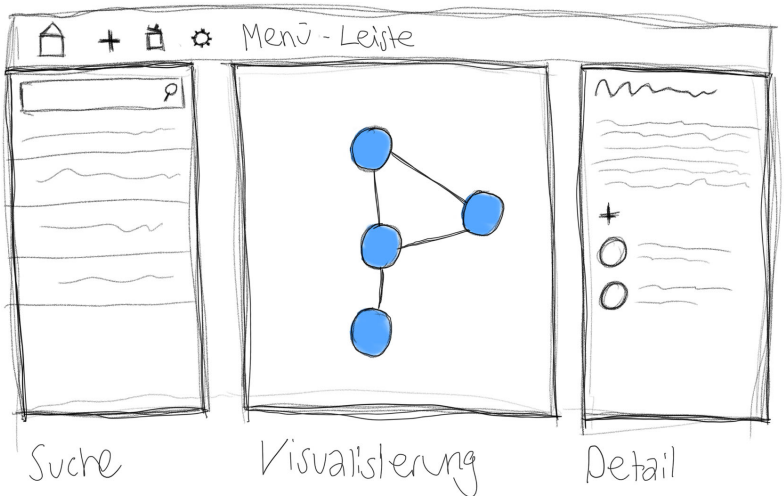


Abbildung 3.7: 3-spaltige Oberfläche

kann als Nutzungsvertrag zwischen der Visualisierung und der Komponente, welche sie verwendet, verstanden werden. Beispielsweise interessiert es die Visualisierung wenig, wie und wo ein ***Node*** gespeichert wird. Es wird nur die jeweilige Methode ausgeführt, alles andere geschieht ausserhalb der Visualisierung. Mehr Details dazu sind im Abschnitt 3.3 zu finden.

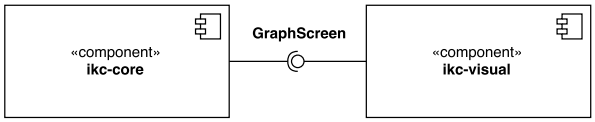


Abbildung 3.8: Komponentendiagramm

3.3 Architektur

Das Klassendiagramm (Abbildung 3.9) gewährt einen detaillierten Überblick über die Architektur der Visualisierung. Es zeigt die Schnittstellen zum ***ikc-core*** und die wichtigsten Komponenten auf. Die *React*-Klasse *GraphScreen* repräsentiert das gesamte Packet ausserhalb der Visualisierung. Sie hält alle internen Klassen, Interfaces und Komponenten zusammen und stellt sicher, dass Informationen zum richtigen Zeitpunkt bei der richtigen Klasse platziert werden. *Graph* als zweite grosse *React*-Klasse kapselt das ***Framework*** *cytoscape* und stellt die Interaktion mit der Visualisierung sicher. Die weiteren Schnittstellen, Klassen und ihre Implementationen werden im Abschnitt 3.4, Abschnitt 3.5 und Kapitel 4 genauer ausgeführt.

Abbildung 3.9: Klassendiagramm

3.4 Schnittstellen

Die hier gezeigten Schnittstellen dienen dem Austausch und der Interaktion mit dem *ikc-core*. Um die Visualisierung in einer bestehenden Umgebung zu verwenden, müssen die erforderlichen Teile implementiert werden.

Die Abbildung 3.10 zeigt einen Teil einer möglichen Integration, beispielsweise in den bestehenden ***ikc-core***. Die rechte Seite stellt einen Ausschnitt des *ikc-visual*-Paketes an. Die Klasse *GraphScreen* (Unterabschnitt 3.4.1) ist zuständig für die Visualisierung des ***Netzwerks***. Um den vollen Funktionsumfang zu bieten, benutzt sie, unter anderem, die Schnittstelle *DialogFactory* (Unterabschnitt 3.4.5). Diese soll den Umgang mit Dialogfenstern ermöglichen. Da dies zu den grundlegenden Funktionen der Visualisierung zählt, befindet sich diese Schnittstelle direkt im *ikc-visual*-Paket. Bei der Integration der Visualisierung in eine bestehende Umgebung gilt es somit, alle Schnittstellen entsprechend zu implementieren.

Die Klasse *GraphVisualisation* verwendet die Klasse *GraphScreen* aus der Visualisierung. Folglich muss beispielsweise die Schnittstelle *DialogFactory* implementiert werden. Dies wird mit der Klasse *GraphDialogFactory* realisiert.

Ähnlich wie beim oben beschriebenen Beispiel gibt es weitere Voraussetzungen des *GraphScreen*, welche bei einer Integration beachtet werden sollten. Diese werden nachfolgend genau erläutert.

Es folgt eine Beschreibung der im *ikc-visual*-Paket enthaltenen Klassen und Schnittstellen.

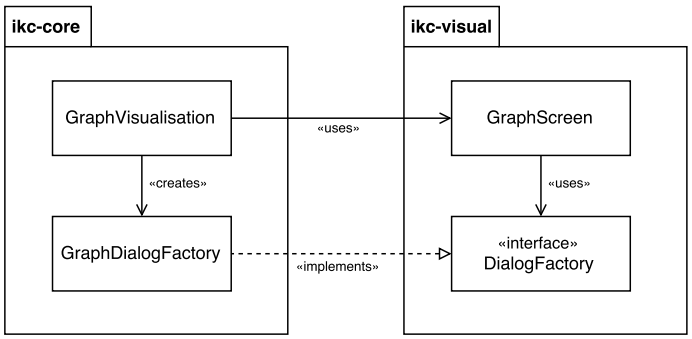


Abbildung 3.10: Integration ***ikc-core***

3.4.1 GraphScreenProps

Die *React*-Komponente *GraphScreen* wird seitens der Visualisierung implementiert. Sie ist die einzige, welche ausserhalb der Visualisierung genutzt werden kann. Damit ist deren Nutzung gleichzustellen mit der Nutzung der Visualisierung. Mit Hilfe der Schnittstelle *GraphScreenProps* wird geregelt, wie die Komponente genutzt werden kann und welche Informationen, respektive welche Implementationen ihr zur Verfügung gestellt werden müssen.

Somit sammelt die *GraphScreen*-Komponente alle Abhängigkeiten und repräsentiert das ***Netzwerk*** als Ganzes. Sie hält alle Referenzen zu Daten, Funktionen, sowie zu den umliegenden Komponenten. Damit stellt sie das einzige Bindeglied zwischen ***ikc-core*** und der Visualisierung dar. Für die Verwendung der Komponente sind die folgenden Objekte bereitzustellen: Ein grosser Teil in Form von Implementationen von Schnittstellen, welche in der Visualisierung definiert sind. Diese werden in den nächsten Abschnitten genauer erläutert (Code Ausschnitt 1):

- *viewToLoad* - View, welche angezeigt wird.
- *nodeInformationProvider* - Stellt weitere Informationen zu ***Nodes*** und ***Links*** zur Verfügung.
- *operationService* - Ermöglicht Interaktion mit der Datenbasis.
- *timestamp* - Timestamp der letzten Verwendung.
- *dialogFactory* - Ermöglicht den Zugriff auf die verschiedenen Dialoge.
- *searchFieldFactory* - Ermöglicht den Zugriff auf die verschiedenen Suchfelder.
- *onNodeDetailRequest* - ***Callback***-Methode, wenn weitere Informationen zu einem ***Node*** gewünscht sind.
- *nodeTypes* - Eine Liste verschiedener Arten von ***Nodes***, z.B. *Plain*, *Dropbox*, etc.
- *identityService* - Ermöglicht das Erstellen von eindeutigen Indentifikationsnummern für neu erstellte ***Nodes*** und ***Links***

In Code Ausschnitt 2 ist ein Beispiel für die Verwendung aufgezeigt.

3.4.2 NodeInformationProvider

Die Visualisierung benötigt für die Darstellung des ***Netzwerks*** diverse Informationen zu den einzelnen ***Nodes*** und ***Links***. Dazu bietet

```
export interface  GraphScreenProps {
    viewToLoad: View;
    nodeInformationProvider: NodeInformationProvider;
    operationService: OperationService;
    timestamp: string;
    dialogFactory: DialogFactory;
    searchFieldFactory: SearchFieldFactory;
    onNodeDetailRequest: Function;
    nodeTypes: GraphNodeType[];
    identityService: IdentityService;
}
```

Listing 1: GraphScreen-Komponente

der *NodeInformationProvider* die beiden Methoden `getNodeTitle` und `getLinkLabel`. Die Identifikation erfolgt jeweils über eine eindeutige Identifikationsnummer (ID) (Code Ausschnitt 3).

Teile der Visualisierung sollen mittels ***Drag'n'Drop*** bedient werden können. Folgend ein Beispiel, wie die Schnittstelle in diesem Anwendungsfall benutzt werden kann:

Für ein *Drag and Drop* wird üblicherweise ein ***Event*** beim Loslassen des gepackten Elements ausgelöst. Das Element soll eine eindeutige ID enthalten. Dies ist aus dem bestehenden *IKC-Core* vorausgesetzt. Wird das Element nun innerhalb der Visualisierung losgelassen, enthält der ausgelöste ***Event*** diese ID. Sie wird zusammen mit dem ***Event*** abgefangen und kann weiterverwendet werden. Beispielsweise können nun Informationen über den *NodeInformationProvider* bezogen werden.

Diese Schnittstelle deckt alle Leseoperationen ab. Schreiboperationen sind in der Schnittstelle *OperationService* (Code Ausschnitt 4) zusammengefasst und werden im nächsten Abschnitt ausgeführt.

3.4.3 OperationService

Die *OperationService*-Schnittstelle ermöglicht der Visualisierung Operationen an die Datenbasis weiterzuleiten. Damit spezifiziert und bündelt diese Schnittstelle alle Schreiboperationen, welche vom ***ikc-core***

```
let viewToLoad: View = new View(...);

let nodeTypes: GraphNodeType[] = [...];

let timestamp: string = ...;

let onNodeDetailRequested: Function = ((nodeId:string) => {
   ...
});


let nodeInformationProvider: NodeInformationProvider =
    new NodeInformationProviderImpl(...);

let operationService: OperationService =
    new OperationServiceImpl(...);

let dialogFactory: DialogFactory =
    new DialogFactoryImpl(...);

let searchFieldFactory: SearchFieldFactory =
    new SearchFieldFactoryImpl(...);

let identityService: IdentityService =
    new IdentityServiceImpl(...);

<GraphScreen
↪   nodeInformationProvider={nodeInformationProvider}
              operationService={operationService}
              viewToLoad={viewToLoad}
              timestamp={timestamp}
              dialogFactory=dialogFactory}
              searchFieldFactory={searchFieldFactory}
              onNodeDetailRequest={onNodeDetailRequested}
              nodeTypes={nodeTypes}
              identityService={identityService}
/>

```


Listing 2: GraphScreen-Beispiel

```
1  /**
2   * Provides Information for visualization => connection to
   ↪  specific data structure. This has to be implemented to
   ↪  use this package.
3   */
4  export interface NodeInformationProvider {
5      getNodeTitle(id: string): string;
6      getLinkLabel(sourceId: string, linkId: string): string;
7  }
```

Listing 3: NodeInformationProvider-Interface

implementiert werden müssen (Code Ausschnitt 4). Diese Operationen entsprechen den folgenden vier Methoden:

1. `createNodeFromDialogState` - neuer ***Node*** auf der Basis des *NewNode* Dialogs.
2. `createNodeWithLinkFromDialogState` - neuer ***Node***, zusammen mit einem ***Link*** zu einem bestehenden ***Node***, auf der Basis des *NewNodeToConnect* Dialogs.
3. `createLink` - neuer ***Link*** definiert durch *id*, den beiden ***Nodes*** *source* und *target* und dem entsprechenden *label*.
4. `saveView` - speichert die übergebene ***View***.

3.4.4 IdentityService

Jeder ***Node*** und jeder ***Link*** in der Visualisierung als auch in der zugrundeliegenden Datenbasis verfügt über eine ID. Diese ermöglicht es, das Element eindeutig zu identifizieren und Operationen korrekt auszuführen. So kann ein konsistenter Kontext gewährleistet werden. Neue Elemente, welche aus der Visualisierung entstehen, müssen ebenfalls mit einer korrekten ID versehen werden. Dazu bietet die Schnittstelle *IdentityService* (Code Ausschnitt 5) die beiden Methoden `createNewNodeId` und `createNewLinkId`.

3.4.5 DialogFactory

Durch die Entkopplung der Erstellung von Dialogen und der Visualisierung ist es möglich, die Visualisierung sehr flexibel einzusetzen.

```
1  /**
2   * Process certain events in the specific data structure.
   ↪  This has to be implemented to use this package.
3   */
4  export interface OperationService {
5      createNodeFromDialogState(state: DialogNewNodeState):
   ↪  void
6      createNodeWithLinkFromDialogState(state:
   ↪  DialogNewNodeToConnectState): void
7      createLink(id: string, source: string, target: string,
   ↪  label: string): void;
8      saveView(view: View): void;
9  }
```

Listing 4: OperationService-Interface

```
1  /**
2   * Provides methods to create ids for new nodes and links.
   ↪  This has to be implemented to use this package.
3   */
4  export interface IdentityService {
5      createNewNodeId(): string
6      createNewLinkId(): string
7  }
```

Listing 5: IdentityService-Interface

Die Visualisierung definiert zwar, welche Dialoge existieren, deren Implementation wird aber komplett ausgelagert. Diese Implementation ist für die Visualisierung nur soweit bedeutsam, als dass sie die benötigten Informationen liefert. Die Dialog-Schnittstellen definieren die dazu nötigen Eigenschaften. Diese Spezifikationen werden dann vom ***ikc-core*** für die Implementation verwendet. Die verschiedenen Schnittstellen werden im nächsten Unterabschnitt 3.4.6 detailliert behandelt.

Mit Hilfe der Implementierung des *DialogFactory*-Interfaces kann die Visualisierung die entsprechenden Dialoge erstellen und mit den nötigen ***Callback*** Methoden ausstatten. Dies funktioniert auch im Falle, dass die Implementation für die Visualisierung gänzlich unbekannt ist. Dabei handelt es sich um drei Dialoge, welche über separate Methoden adressiert werden können:

1. `getDialogNodeNewNode` - liefert einen *NewNode*-Dialog.
2. `getDialogNodeNewNodeToConnect` - liefert einen *NewNodeToConnect*-Dialog.
3. `getDialogNodeSearchToConnect` - liefert einen *SearchNodeToConnect*-Dialog.

3.4.6 Dialog-Schnittstellen

Wie bereits aufgezeigt, übernehmen die Dialog Schnittstellen die Aufgabe, die minimalen Eigenschaften der verschiedenen Dialoge zu spezifizieren. Dies wird erreicht indem diese Schnittstellen als *State*- und *Props*-Interface verwendet werden. Bei Bedarf können diese erweitert werden, um sie dem Kontext entsprechen anzupassen. Auf diese Weise wird erreicht, dass die Visualisierung die benötigten Informationen erhält. Diese Informationen nutzt sie danach, um die resultierenden Operationen der Dialoge auszuführen.

NewNodeDialog

Mittels des *NewNode*-Dialogs wird es ermöglicht im ***ikc-core*** einen Dialog zu nutzen, welcher dann seine Resultate direkt an die Visualisierung liefert. Die Schnittstelle *DialogNewNodeProps* fungiert dabei als *Props*-Interface. Mittels der Eigenschaft `type` wird der entsprechende Typ des zu erstellenden ***Nodes*** übergeben. Als Datencontainer (`node`) wird die Klasse *GraphNodeElement* verwendet. Über die beiden ***Callback***-Methoden `onRequestClose` und `onSave` wird die Visualisierung über die Endbedingungen des Dialogs informiert. In der

```
1  /**
2   * Specification for the dialog factory which has to
   ↪  implement to use this package
3   */
4  export interface DialogFactory {
5
6          getDialogNodeNewNode(
7                  open: boolean,
8                  onSave: Function,
9                  onRequestClose: Function,
10                 node: GraphNodeElement,
11                 type: GraphNodeType
12         ):React.Component<DialogNewNodeProps,
13                 DialogNewNodeState>
14
15     getDialogNodeNewNodeToConnect(
16                 open: boolean,
17                 onSave: Function,
18                 onRequestClose: Function,
19                 link: GraphLinkElement,
20                 type: GraphNodeType
21         ):React.Component<DialogNewNodeToConnectProps,
22                 DialogNewNodeToConnectState>
23
24     getDialogNodeSearchToConnect(
25                 open: boolean,
26                 onSelect: Function,
27                 onRequestClose: Function
28         ):React.Component<DialogNodeSearchToConnectProps,
29                 DialogNodeSearchToConnectState>
30 }
```

Listing 6: DialogFactory-Interface

Schnittstelle `DialogNewNodeState` wird der aktuelle Status des Dialogs gespeichert und schliesslich mit der Methode `onSave` an die Visualisierung übergeben. Dort kann der Status weiterverarbeitet werden.

```
1  /**
2   * State and props interface for the "new node" dialog.
   ↪  Which has to be implemented to use this package
3   */
4  export interface DialogNewNodeProps {
5      open: boolean
6      timestamp: string
7      node: GraphNodeElement
8      onSave: Function
9      onRequestClose: Function
10     type: GraphNodeType
11 }
12
13 export interface DialogNewNodeState {
14     timestamp?: string
15     node?: GraphNodeElement
16 }
```

Listing 7: DialogNewNode-Schnittstellen

NewNodeToConnectDialog

Nebst der Erstellung eines ***Nodes*** soll es auch ermöglicht werden, mit Hilfe eines Dialogs einen neuen ***Node*** zu erstellen, welcher mittels eines ***Links*** mit einem bestehenden ***Node*** verbunden ist. Hierzu werden zwei zusätzliche Schnittstellen definiert: *DialogNewNodeToConnectProps* und *DialogNewNodeToConnectState*. Im Gegensatz zu den Schnittstellen des *NewNodeDialog* wird hier als Datencontainer (`link`) für die Visualisierung die Klasse *GraphLinkElement* verwendet.

DialogNodeSearchToConnectInterfaces

Der dritte Dialog dient dazu, einen ***Node*** aus der bestehenden Datenbasis zu suchen und diesen mit einem existierenden ***Node*** in der

```
1  /**
2   * State and props interface for the "new node and connect
   ↪  to an existing" dialog. Which has to be implemented to
   ↪  use this package
3   */
4  export interface DialogNewNodeToConnectProps {
5      open: boolean,
6      timestamp: string
7      link: GraphLinkElement
8      onSave: Function
9      onRequestClose: Function
10     type: GraphNodeType
11 }
12
13 export interface DialogNewNodeToConnectState {
14     timestamp?: string
15     link?: GraphLinkElement
16 }
```

Listing 8: DialogNewNodeToConnect-Interfaces

Visualisierung mittels eines ***Link*** zu verknüpfen. Wiederum werden die Eigenschaften dieses Dialogs mithilfe zweier Schnittstellen definiert: *DialogNodeSearchToConnectProps* stellt wiederum sicher, dass die benötigten ***Callback***-Methoden zur Verfügung stehen. *DialogNodeSearchToConnectState* sorgt dafür, dass die richtigen Informationen an die Visualisierung übergeben werden.

3.4.7 SearchFieldFactory

Sowohl das *CoreContextMenu* als auch das *NodeContextMenu* sind Komponenten, welche die Visualisierung liefert. Die integrierte Suche kann jedoch nicht von ihr zur Verfügung gestellt werden, da sie keinen Zugriff auf die zugrundeliegende Datenbasis hat. Dazu wird die *SearchFieldFactory* Schnittstelle verwendet, welche Zugriffe auf das *NodeSearchField* und das *LinkSearchField* regelt. Auch hier wird die konkrete Implementation der Suchkomponenten im ***ikc-core*** erledigt.

Die einzelnen Komponenten werden über die Methoden `getNode-`

```
/**
 * State and props interface for the "search node and
 ↪ connect to an exsting node" dialog. Which has to be
 ↪ implemented to use this package
 */
export interface DialogNodeSearchToConnectProps {
    open: boolean,
    timestamp: string
    onSave: Function
    onRequestClose: Function
    title: string
}

export interface DialogNodeSearchToConnectState {
    nodeName?: string
    timestamp?: string
    link?: GraphLinkData
}
```

Listing 9: DialogSearchNodeToConnect-Interfaces

`SearchField` und `getLinkSearchField` geliefert. Dazu muss bei beiden eine ***Callback***-Methode übergeben werden, welche bei der Auswahl eines Suchresultats ausgeführt wird. Bei der Methode `getLinkSearchField` muss zusätzlich ein ***Array*** von den Link-IDs mitgeliefert werden. Darüber werden die Elemente festgelegt, welche bei der Suche berücksichtigt werden sollen.

3.5 Model

Klassen oder Schnittstellen, welche sich auf die Haltung von Daten beschränken, werden als *Model* zusammengefasst. So repräsentieren sie zum Beispiel eine Koordinate, einen Knoten oder einen Link. Da sie kein konkretes Verhalten implementieren, werden sie bereits im Lösungsdesign definiert. In anderen Sprachen, wie zum Beispiel ***Scala***, werden sie auch als sogenannte *Case Classes* bezeichnet ((ScalaLang, 2017)).

```
/**
 * Specification for the search field factory which has to
 ↪  implement to use this package
 */
export interface SearchFieldFactory {
    getNodeSearchField(onNodeSelected: Function):
 ↪  React.Component<any,any>
    getLinkSearchField(onLinkSelected: Function, links:
 ↪  string[]): React.Component<any,any>
}
```

Listing 10: SearchFieldFactory-Schnittstelle

GraphElement

Die Schnittstelle *GraphElement* ist das grundlegende Interface von ***Nodes*** und ***Links***. Es wird durch zwei Eigenschaften spezifiziert (Code Ausschnitt 11):

- `data` - enthält die nötigen Informationen zum jeweiligen Element.
- `visibility` - beschreibt die Sichtbarkeit des Elements.

```
/**
 * Interface for all graph elements
 */
export interface GraphElement {
    data: GraphElementData
    visibility: VISIBILITY
}
```

Listing 11: GraphElement-Schnittstelle

GraphElementData

Mit der einzigen Eigenschaft `id`, sorgt die Schnittstelle *GraphElement-Data* für eine eindeutige Identifikation aller Elemente (Code Ausschnitt 12).

```
/**
 * Interface for all graph element data
 */
export interface GraphElementData {
        id: string
}
```

Listing 12: GraphElementData-Interface

GraphNodeElement

Die Klasse *GraphNodeElement* repräsentiert einen einzelnen ***Node*** und implementiert die Schnittstelle *GraphElement*. Diese wird mit der Eigenschaft `position` erweitert, welche die aktuelle Position innerhalb der Visualisierung enthält und `nodeClasses`, um potentielle Node-Klassen zu speichern. Ebenfalls nutzt diese Klasse die entsprechende *GraphElementData* Implementierung *GraphNodeData* . Ein *GraphNodeElement* hat immer auch die gleiche ID und Titel wie in der Datenbasis, die restlichen Eigenschaften unterscheiden sich jedoch (Code Ausschnitt 13).

```
/**
 * Graph data element specify the graph element
 */
export class GraphNodeElement implements GraphElement {
    data: GraphNodeData
    position: GraphPosition
    visibility: VISIBILITY
    nodeClasses: any[]
}
```

Listing 13: GraphNodeElement-Klasse

GraphNodeData

Die ***Node*** spezifischen Daten werden in der Klasse *GraphNodeData* gehalten. Diese entspricht einer konkreten Implementierung der Schnittstelle *GraphElementData*. Ergänzt wird sie durch die Eigenschaft `title`,

welche den Titel des ***Nodes*** enthält (Code Ausschnitt 14).

```
/**
 * Graph node data specify the graph element data
 */
export class GraphNodeData implements GraphElementData {
    id: string
    label: string

}
```

Listing 14: GraphNodeData-Klasse

GraphLinkElement

Auch für die Repräsentation eines ***Links*** wird eine separate Implementation *GraphLinkElement* der Schnittstelle *GraphElement* verwendet. Mit der Eigenschaft `linkClasses` speichert sie eventuelle Link-Klassen. Auch hier wird zusätzlich eine Implementation *GraphLinkData* der Schnittstelle *GraphElementData* für das `data` Objekt verwendet (Code Ausschnitt 15).

```
/**
 * Graph link element specify the graph element
 */
export class GraphLinkElement implements GraphElement {
    data: GraphLinkData
    visibility: VISIBILITY
    linkClasses: string[]
}
```

Listing 15: GraphLinkElement-Klasse

GraphLinkData

Auch für die ***Link*** spezifischen Daten wird eine Implementation *GraphLinkData* der Schnittstelle *GraphElementData* verwendet. Diese wird ergänzt durch die jeweiligen IDs `source` und `target` der ***Nodes***, wel-

che den ***Link*** mit den jeweiligen ***Nodes*** verbindet und ein entsprechendes Label (`label`) (Code Ausschnitt 16).

```
/**
 * Graph link data specify the graph element data
 */
export class GraphLinkData implements GraphElementData {
    id: string
    source: string
    target: string
    label: string
}
```

Listing 16: GraphLinkData-Klasse

GraphPosition

Um eine Position in der Visualisierung zu persistieren, werden die X- und Y-Koordinaten gespeichert. Dazu wird die Klasse *GraphPosition* verwendet (Code Ausschnitt 17).

```
/**
 * Single position with x and y coordinates
 */
export class GraphPosition {
    x: number
    y: number
}
```

Listing 17: GraphPosition-Klasse

View

Um die verschiedenen ***View***s und die Positionen der enthaltenen ***Nodes*** zu persistieren, wird die Klasse ***View*** verwendet. Sie enthält dabei die grundlegende Struktur des gesamten ***Netzwerks***. Mit Hilfe des Enumerators VISIBILITY (Abschnitt 3.5) wird festgelegt, welche

Links und ***Nodes*** tatsächlich auch dargestellt werden. Dazu hat sie die folgenden Eigenschaften: (Code Ausschnitt 18):

- `id` - eindeutige Identifikation.
- `titel` - Titel des ***Nodes***.
- `changedAt` - Timestamp der letzten Aktualisierung.
- `createdAt` - Timestamp der Erstellung.
- `nodes` - Liste aller ***Nodes*** der Datenbasis. In der Visualisierung werden die einzelnen ***Nodes*** abhängig von der Eigenschaft `visibility` dargestellt.
- `links` - Liste aller ***Links*** der Datenbasis. Auch hier werden die ***Links*** anhand der Eigenschaft `visibility` dargestellt oder nicht.

```
1  /**
2   * Save a visualisation
3   */
4  export class View {
5      id: string
6      title: string
7      changedAt: string
8      createdAt: string
9      nodes: GraphNodeElement[]
10     links: GraphLinkElement[]
11 }
```

Listing 18: View-Klasse

VISIBILITY

Mit dem Enumerator *VISIBILITY* wird die Sichtbarkeit eines Elements festgelegt. Die ist entweder `VISIBLE` oder `HIDDEN` (Code Ausschnitt 18). Da *Typescript* keine String-Enumeratoren kennt, wird stattdessen eine normale Klasse verwendet.

3.6 Framework-Auswahl

Um sicherzustellen, dass ein geeignetes ***Framework*** für die Visualisierung verwendet wird, müssen verschiedene Optionen untersucht

```
/**
 * Enumerator for the visibility flag of nodes and links
 */
export class VISIBILITY {
    toString() {
        return this.value;
    }

    static VISIBLE = new VISIBILITY("VISIBLE");
    static HIDDEN = new VISIBILITY("HIDDEN");
}
```

Listing 19: VISIBILITY-Enumerator

und bewertet werden.

3.6.1 Kriterien-Katalog

Für die Bewertung wird ein Katalog an Kriterien definiert, welcher sich an den definierten Anforderungen (siehe Abschnitt 2.6), der Risikoanalyse (siehe Abschnitt 2.7), den Schnittstellen (siehe Abschnitt 3.4) und den Komponenten (siehe Unterabschnitt 3.1.2) orientiert. Folgend eine Übersicht über den Kriterienkatalog (Punkt 3.2).

ID	Titel	Beschreibung
K1	Mobile Darstellung	Das ***Framework*** kann im mobilen Umfeld verwendet werde
K2	Operationen	Es können die folgenden Operationen umgesetzt werden: 1. Einen neuen ***Node*** innerhalb der Visualisierung erstellen. 2. Zwei ***Nodes*** innerhalb der Visualisierung verbinden. 3. Einen ***Node*** innerhalb der Visualisierung löschen.

K3	Operationen (***Drag'n'Drop***)	Es können die folgenden ***Drag'n'Drop***-Operationen umgesetzt werden: 1. Einen neuen ***Node*** mittels ***Drag'n'Drop*** innerhalb der Visualisierung erstellen. 2. Zwei ***Nodes*** innerhalb der Visualisierung mittels ***Drag'n'Drop*** verbinden.
K4	***Node*** Details	***Node*** Details können angezeigt und bearbeitet werden.
K5	Menü	Ein Kontextmenü für weitere Operationen anzeigen.
K6	***Nodes*** Speichern	Positionen der ***Nodes*** können gespeichert werden.
K7	***Nodes*** Laden	Eine bestehende View kann dargestellt werden.
K8	***Toolbox***	Weitere Operationen, z.B. eine Suche, können in einer ***Toolbox*** angeboten werden.
K9	Komplexität	Allgemeiner Eindruck des ***Framework***s hinsichtlich der Komplexität.
K10	Erweiterbarkeit	Allgemeiner Eindruck des ***Framework***s hinsichtlich der Erweiterbarkeit.
K11	Dokumentation	Das ***Framework*** ist gut dokumentiert, es gibt genügend Beispiele und eine entsprechende *Community* zur allfälligen Unterstützung.

Tabelle 3.2: Kriterienkatalog

Mit Hilfe dieses Katalogs sollen die Möglichkeiten, Chancen und auch Risiken der verschiedenen ***Framework***s identifiziert werden. Aufgrund dieses Prozesses kann anschliessend eine genaue Einschätzung der Möglichkeiten und des Aufwands aufgestellt werden. Die verschiedenen ***Framework***s werden anhand der folgenden Skala bewertet (1-5):

1. Die Erfüllung des Kriteriums **ohne** Anpassungen möglich.
2. Die Erfüllung des Kriteriums mit **leichten** Anpassungen möglich.
3. Die Erfüllung des Kriteriums ist **mit Anpassungen** möglich.

4. Die Erfüllung des Kriteriums ist mit **grossen** Anpassungen möglich.
5. Die Erfüllung des Kriteriums ist **nicht** möglich.

3.6.2 Ergebnisse

Aufgrund des definierten Kriterienkataloges wurden die verschiedenen ***Framework***s untersucht und bewertet. Die Ergebnisse sind in der folgenden Tabelle 3.3 aufgeführt. Die Tabelle widerspiegelt die Eindrücke und Erfahrungen, welche während den Untersuchungen gemacht wurden:

Zwar sind auch die umfassenderen Lösungen sehr interessant, jedoch ist deren Verwendung für den hier notwendigen Zweck zu kompliziert. Es würde lediglich ein kleiner Teilbereich der bereits bestehenden Lösung genutzt. Um aber diesen erfolgreich einzusetzen, ist dennoch tiefe Kenntnis des jeweiligen ***Framework***s erforderlich. Dies sprengt schlichtweg den zeitlichen Rahmen und ist auch nicht notwendig. Die nötigen Erweiterungen können in den anderen, schlankeren ***Framework***s ohne grossen Zusatzaufwand ergänzt werden.

Darum fiel die Wahl eindeutig auf *cytoscape.js*. Es beschränkt sich lediglich auf die Darstellung von ***Netzwerken***. Diverse Erweiterungen sind bereits zugänglich. Zudem ist es gleichzeitig relativ einfach den Funktionsumfang eigenhändig zu ergänzen.

	cytoscape.js	*greuler*	*JointJS*	*jsPlumb*	*mxGraph*
Total	23	27	38	24	27

Tabelle 3.3: Framework-Auswerung

3.6.3 Bewertungen

Eine detaillierte Bewertung der fünf verschiedenen ***Framework***s kann den folgenden Abschnitten entnommen werden.

cytoscape.js

Cytoscape ist eine *open-source* Javascript Bibliothek für Graphen- oder ***Netzwerk***-Theorie. Sie eignet sich nicht nur für Visualisierungen, sondern auch für Analysen. Cytoscape ist mit allen gängigen Browsern und Bibliotheken kompatibel. Zudem funktioniert die Darstellung ohne zusätzlichen Aufwand auf allen Bildschirmgrössen. Es gibt zahlrei-

che Erweiterungen und auch die Integration in bestehende Lösungen ist einfach möglich. (Franz et al., 2016)

Wie in Abbildung 3.11 ersichtlich, beschränkt sich die Bibliothek auf die Visualisierung von ***Netzwerken***. Standardmässig sind keine Zusatzfunktionen, beispielsweise das Interagieren mit ***Nodes*** und ***Links*** möglich. Die Bibliothek ist sehr schlank gehalten, was eine einfache Integration und Erweiterung stark vereinfacht. Das ***Netzwerk*** wird direkt im *JSON*-Format hinterlegt. Bei Änderungen werden die hinterlegten Daten angepasst und anschliessend angezeigt.

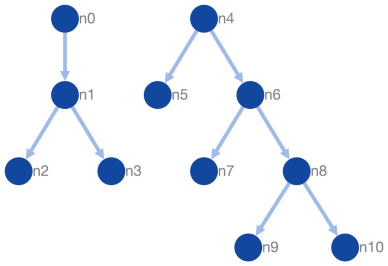


Abbildung 3.11: Beispiel Cytoscape

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
2	2	2	2	2	2	3	2	2	3	1

Tabelle 3.4: Bewertung *cytoscape.js*

greuler

Hier gibt es viele Parallelen zur vorgängigen Bibliothek (*cytoscape.js*). *Greuler* spezialisiert sich nun aber auf die Visualisierung und baut wie viele andere auf *D3*. Bezüglich der Handhabung ist es grösstenteils identisch mit *Cytoscape*, allerdings hat es hier bei weiten nicht so viele Erweiterungsmöglichkeiten. Abbildung 3.12 zeigt ein kleines Beispielnetzwerk. (Poppe, 2016)

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
2	2	2	2	2	4	4	2	2	4	3

Tabelle 3.5: Bewertung *greuler*

JointJS

Auch bei JointsJS handelt es sich um eine *open-source*-Bibliothek. Im Gegensatz zu den obigen beiden geht die Funktionalität eine Ebe-

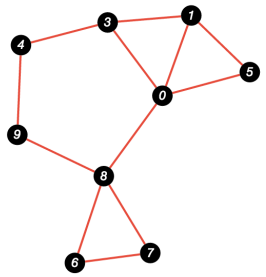


Abbildung 3.12: Beispiel Greuler

ne weiter. Hier werden zusätzlich zur eigentlichen Darstellung auch Werkzeuge zur Manipulation des Diagramms mitgeliefert. Viele Funktionalitäten sind aber erst mit der kostenpflichten Version *Rappid* verfügbar, welche auf JointJS aufbaut. (Rappid, 2016)

Die Bibliothek ist für den hier vorliegenden Anwendungsfall zu umfangreich. Die vielen Funktionen machen die Implementation sehr komplex. Dieser Aufwand ist für die eigentlich einfache Anwendung zu gross. Die Vorteile, welche der vorhandene Funktionsumfang bietet, erkennt man erst nach längerer Einarbeitung. Der notwendige Funktionsumfang kann in anderem ***Frameworks*** ohne grossen Aufwand selbst implementiert werden. So ist die Übersicht stets geboten und der Code bleibt verhältnismässig einfach.

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
1	3	3	3	3	4	4	5	4	5	3

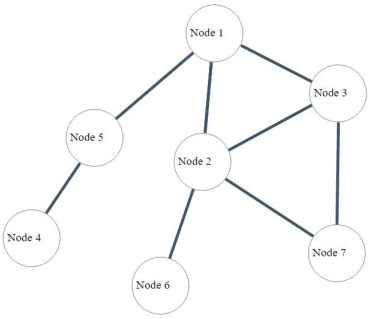
Tabelle 3.6: Bewertung *JointJS*

jsPlumb

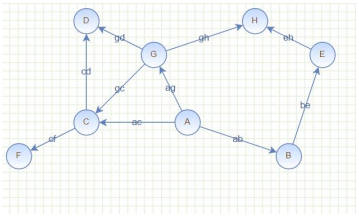
K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
2	2	2	1	3	2	2	2	3	3	2

Tabelle 3.7: Bewertung *jsPlumb*

Als einzige Bibliothek basiert jsPlump auf reinen ***HTML***-Objekten. Somit werden diese nicht in einem ***Canvas*** gezeichnet, sondern wie normaler ***HTML*** Code dargestellt. Somit kann das Aussehen der Elemente direkt durch ***CSS***-Klassen angepasst werden.



(a) Beispiel JsPlump



(b) Beispiel mxGraph

Abbildung 3.13: Frameworks

Jedoch legt jsPlump den Fokus vor allem auf das Zeichnen und Erstellen von Diagrammen. Es bietet daher eine grosse Anzahl von Funktionen, welche nicht benötigt werden. Zusätzlich müssten benötigte Funktionen zum Teil manuell implementiert werden. Ebenfalls ist nur die limitierte *Community Edition* kostenlos verfügbar. (jsplump, 2017)

mxGraph

Wie auch schon bei der Bibliothek jsPlump setzt mxGraph den Fokus auf die Darstellung und Bearbeitung von ganzen Diagrammen. Es wurde schon als Grundlage von grossen Anwendung wie *draw.io* verwendet. Dadurch ist eine Vielzahl von Funktionen verfügbar. Jedoch würde sich eine Erweiterung dieser eher kompliziert gestalten. (mxGraph, 2017)

K1	K2	K3	K4	K5	K6	K7	K8	K9	K10	K11
2	3	3	2	3	2	2	2	4	4	2

Tabelle 3.8: Bewertung *mxGraph*

Kapitel 4

Implementation

Die Implementation hat zum Ziel, die bereits erarbeitete konzeptionelle Lösung in Form von Code praktisch umzusetzen. Nach einer Übersicht über die abgeschlossene Visualisierung und einem Einblick in die eingesetzten Technologien wird im Detail auf nennenswerte Eigenheiten des Projektes eingegangen.

4.1 Übersicht

Nachfolgend ist die integrierte Visualisierung innerhalb des ***ikc-cores*** ersichtlich (Abbildung 4.1). Sie wurde zwischen der Suche und der Detail-Ansicht platziert. Eine detaillierte Anleitung der Interaktion mit der integrierten Visualisierung ist dem Benutzerhandbuch zu entnehmen.

4.2 Technologie

Für die Umsetzung wurde aufgrund vieler Parallelen zum ***ikc-core*** und diverser guten Erfahrungen auf ähnliche Technologien gesetzt. Das Grundgerüst ist daher nahezu identisch mit dem bisherigen. Dabei konnte bereits vorhandenes Wissen wiederverwendet und erweitert werden.

4.2.1 Typescript

Typescript ist eine von Microsoft entwickelte statisch typisierte Erweiterung von Javascript. Durch die Kompilierung wird es zu üblichen Javascript. Typescript kann bereits während des Schreibens überprüft werden und die Entwicklungstools bieten dadurch viel Hilfestellung.

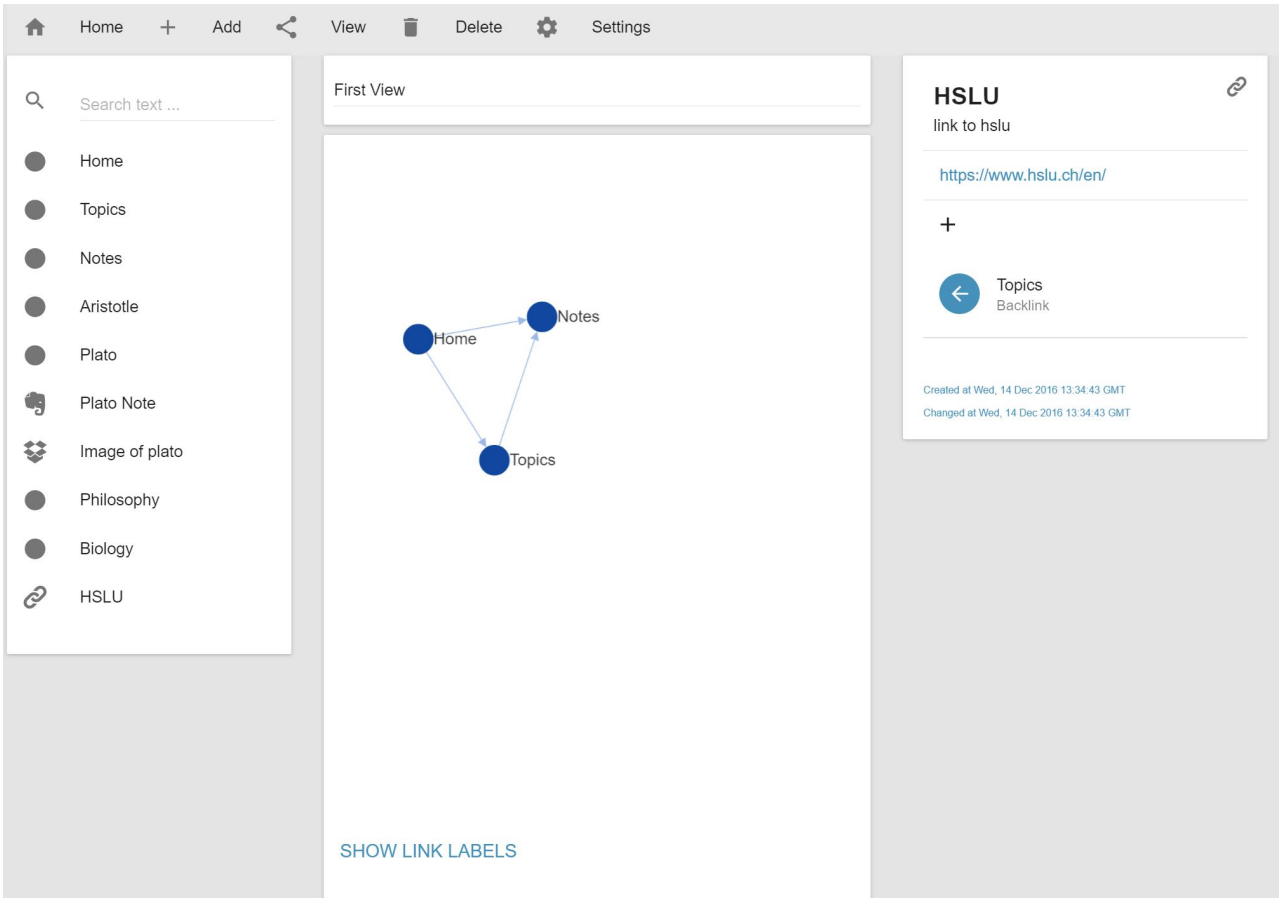


Abbildung 4.1: Überblick Implementation

Die Typen sind zwar optional, falls eingesetzt bringen sie aber einige hilfreiche Zusatzfunktionen. Beispielsweise lassen sich Klassen, Interfaces definieren. Dies ist vor allem bei der Entwicklung von grösseren Projekten hilfreich. (Typescript, 2017)

Typescript wurde vor allem aufgrund der einfacheren Strukturierung von Projekten und der Typisierung eingesetzt. Zwar kann jegliche Javascript-Bibliothek eingebunden werden, aber ohne bereits vorhandene statische Typisierung in Form von *Typings* bringt dies keine grossen Vorteile. Unglücklicherweise sind die Typisierungen noch nicht für alle Bibliotheken verfügbar und darum muss teilweise auf eine weniger elegante Integration ausgewichen werden.

4.2.2 React

React ist eine Bibliothek für die Entwicklung von Benutzeroberflächen, entwickelt von Facebook. Der Fokus wird insbesondere auf die Inter-

aktivität der Komponenten gelegt, deren Zustände ständig wechseln können. Bei Zustandsänderungen aktualisiert React automatisch alle notwendigen Komponenten. Die Komponentenbauweise bringt hohe Modularität und Wiederverwendbarkeit mit sich, untereinander sind die Komponenten entkoppelt. Einmal erstellte Komponenten können an beliebigen Stellen wiederverwendet werden, so zum Beispiel ein Suchfeld. Verwendet werden dies Komponenten immer in der *XML*-Syntax. Grundsätzlich wird eine React-Komponente in verschiedene Teile zerlegt (Code Ausschnitt 20):

1. *Props-Schnittstelle* - definiert, mit welchen Parameter eine Komponente benutzt werden muss. Optionale Parameter werden mit einem Fragezeichen (`?`) gekennzeichnet. Innerhalb der Komponente kann `this.props` auf die Eigenschaften zugreifen. Jedoch können keine Werte aktualisiert werden.
2. *State-Schnittstelle* - spezifiziert den Zustand der Komponente. Diese kann sich abhängig von Ereignissen innerhalb der Komponente aktualisieren. Mit `this.state` kann ein Zugriff stattfinden.
3. *Implementierung* - jede Komponente implementiert die Schnittstelle *React.Component<>*. Einzig die Methode `render` muss implementiert werden. Darin wird der darzustellende Inhalt aus ***HTML***-Elementen oder weiteren React-Komponenten zusammengestellt.
4. *Verwendung* - nach der Implementierung kann jede Komponente von anderen React-Komponenten innerhalb deren `render` Methode verwendet werden.

Jede React-Komponente durchläuft grundlegende Zustände. An diesen kann mit sogenannten *Lifecycle*-Methoden in den jeweiligen Prozess eingegriffen werden. Bei der Implementierung der Visualisierung wurden hauptsächlich folgenden Methoden eingesetzt.

- *Mounting* - hier werden verschiedene Methoden abgearbeitet, bis eine Komponente erstellt und zu einem ***HTML***-Element konvertiert wird.
    1. `constructor`
    2. `componentWillMount`
    3. `render`
    4. `componentDidMount`

```
1  /** (1) */
2  interface ComponentProps{
3      name?: string,
4      onSave: Function
5  }
6
7  /** (2) */
8  interface ComponentState{
9      name: string
10 }
11
12 /** (3) */
13 class Component extends
   ↪ React.Component<ComponentProps,ComponentState>{
14     ...
15     onTextFieldUpdate = ...
16     ...
17     render(){
18         return(
19             <TextField onChange={this.onTextFieldUpdate}
   ↪ value={this.state.name}
20         )
21
22     }
23 }
24
25 /** (4) */
26 ... <Component name={'Max'}
   ↪ onSave={this.handleSave.bind(this)} />...
```

Listing 20: Beispiel React Komponente

- *Updating* - ein Update kann entweder durch eine Aktualisierung des *State* oder der *Props* erfolgen.
  1. `componentWillReceiveProps`
  2. `shouldComponentUpdate`
  3. `componentWillUpdate`
  4. `render`
  5. `componentDidUpdate`
- *Unmounting* - dieser Schritt wird ausgeführt, bevor die Komponente entfernt wird.
  1. `componentWillUnmount`

Ein weiterer, besonderer Punkt von React ist der unidirektionale Datenfluss. Dieser wird nachfolgend im Unterabschnitt 4.2.3 behandelt. (Facebook, 2017; Wheeler, 2016)

4.2.3 Unidirektionaler Datenfluss

Die Verwendung von unidirektionalen Datenflüssen erleichtert die Verwendung von React-Komponenten, welche auch in dem, von Facebook präsentierten, Architektur-Muster *Flux* enthalten sind. Auf eine strikte Verwendung von *Flux* wurde verzichtet, jedoch wurden trotzdem unidirektionale Datenflüsse eingesetzt. Wie diese genau funktionieren, wird anhand des Beispiels in Abbildung 4.2 genauer aufgezeigt:

1. Die *GraphScreen*-Komponente verwendet die *Graph*-Komponente in ihrer `render`-Methode in Form von (`<Graph ... />`).
2. Sobald den *Graph*-Komponente erstellt ist, wird der Zustand anhand der übergebenen Parameter initialisiert.
3. Angestossen von ***Event***s aus dem *cytoscape*-Framework werden ***Callback***-Methoden aufgerufen, um Information an die *GraphScreen*-Komponente zu senden. Die entsprechenden Methoden werden als Parameter an die *Graph*-Komponente übergeben, beispielsweise `this.prop.handleNewLink(...)`
4. Innerhalb den von der *Graph*-Komponente aufgerufenen Methoden in der *GraphScreen*-Komponente wird nun der Zustand der

GraphScreen-Komponente aktualisiert. Dies geschieht mit der Methode `this.setState(...)`, welche von dem React-Framework zur Verfügung gestellt wird. Hier werden die Änderungen weitergegeben. Diese werden im Hintergrund ausgeführt.

5. Sobald der Zustand angepasst wurde, wird die `render`-Methode neu ausgeführt. Dadurch werden die Aktualisierungen der *Graph*-Komponente per Parameter (*Props*) weitergegeben.

6. Durch die Aktualisierung der Parameter wird nun auch die `render` Methode der *Graph*-Komponente neu ausgeführt und der Zustand aktualisiert.

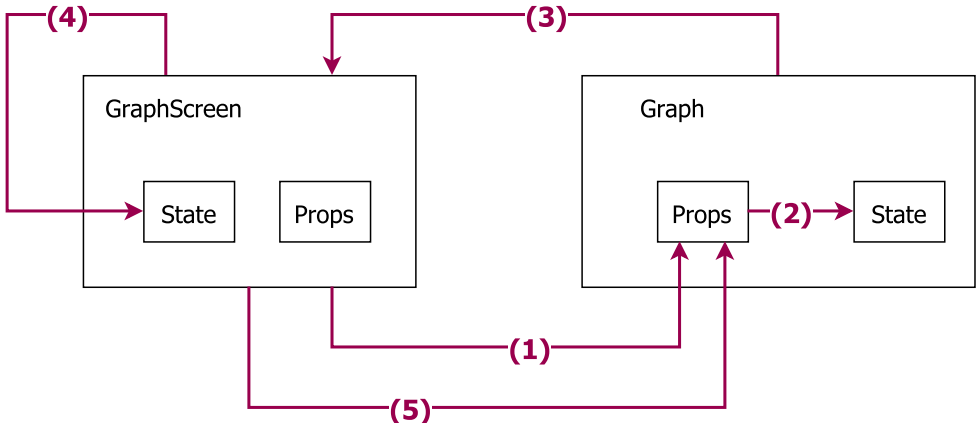


Abbildung 4.2: Unidirektionaler Datenfluss

4.2.4 Material Design

Die React Erweiterung *Material UI* ist zuständig für das Erscheinungsbild der Visualisierung, wie auch schon beim *ikc-core*. Dafür gibt es diverse vorgefertigte Komponenten, welche praktisch ohne Zusatzaufwand verwendet werden können. Googles Material Design kombiniert klassische Design Prinzipien mit Innovation auf Technik und Wissenschaft. (call-em all, 2016; Google, 2016)

4.2.5 Webpack

Mit der Verwendung von zahlreichen Erweiterungen und *Typescript* als primäre Programmiersprache ergeben sich diverse Abhängigkeiten und auch eine Kompilierung ist notwendig. Für die Sammlung der Abhängigkeiten und die Übersetzung in Javascript wird Webpack eingesetzt. Das Resultat ist beispielweise eine Javascript-Datei, wel-

che alle Abhängigkeiten im richtigen Format enthält. Darum ist vom Webserver nur eine Datei zu laden, so verkürzt die Ladezeit zusätzlich. (webpack, 2016)

4.3 Interaktion mit bestehenden Komponenten

Innerhalb der Visualisierung werden verschiedene bestehende Komponenten aus dem ***ikc-core*** genutzt. Diese implementieren die beschriebenen Schnittstellen (Abschnitt 3.4) und werden an den *GraphScreen* übergeben. Die Implementationen gelten als Voraussetzung für die Verwendung der Visualisierung.

GraphDialogFactory/GraphSearchFieldFactory

Mit Hilfe dieser beiden Klassen kann die Visualisierung auf *React*-Komponenten zugreifen, welche im ***ikc-core*** implementiert sind. Diese sind in erster Linie Dialoge oder Suchfelder. Im folgenden Ablaufdiagramm (Abbildung 4.3 und Code Ausschnitt 21) ist die Funktionsweise am Beispiel des *NewNodeDialog*s detailliert ersichtlich. Das Prinzip gilt ebenfalls für die Dialoge *NewNodeToConnect*, *NodeSearchToConnect* und die beiden Suchfelder *NodeSearchField* und *LinkSearchField*:

1. Die Komponente *GraphVisualisation* erstellt ein Objekt der Klasse *GraphDialogFactory*. Sie implementiert die Schnittstelle *DialogFactory* aus der Visualisierung.
2. Bei der Verwendung der Visualisierung (*GraphScreen*) wird unter anderem das *GraphDialogFactory*-Objekt an die Visualisierung übergeben. Dabei ändert das Objekt den Typ von *GraphDialogFactory* zu *DialogFactory*. Da ersteres der Visualisierung unbekannt ist, kann so eine beidseitige Kopplung verhindert werden. Alle Objekte, welche übergeben werden, sind in der Schnittstelle *GraphScreenProps* zusammengefasst. Diese wird innerhalb des *React*-Frameworks implementiert.
3. Wenn nun die Visualisierung einen *NewNode*-Dialog benötigt, wird dieser von der *GraphDialogFactory* bezogen. Diese wird von dem *GraphScreenProps* zur Verfügung gestellt. Dazu wird die Methode `getNewNodeDialog` aufgerufen, welcher die benötigten Informationen für die Erstellung des Dialogs mitgegeben werden.

4. Nachdem der Dialog geschlossen wurde, werden die entsprechenden ***Callback***-Methoden ausgeführt. Diese stehen dem Dialog durch das *DialogNewNodeProps*-Objekt zu Verfügung, welches zuvor von der *GraphDialogFactory* anhand der Informationen der Visualisierung entsprechend bestückt wurde. Somit wird die Visualisierung über das Resultat des Dialogs informiert und kann die Informationen entsprechend weiterverarbeiten.

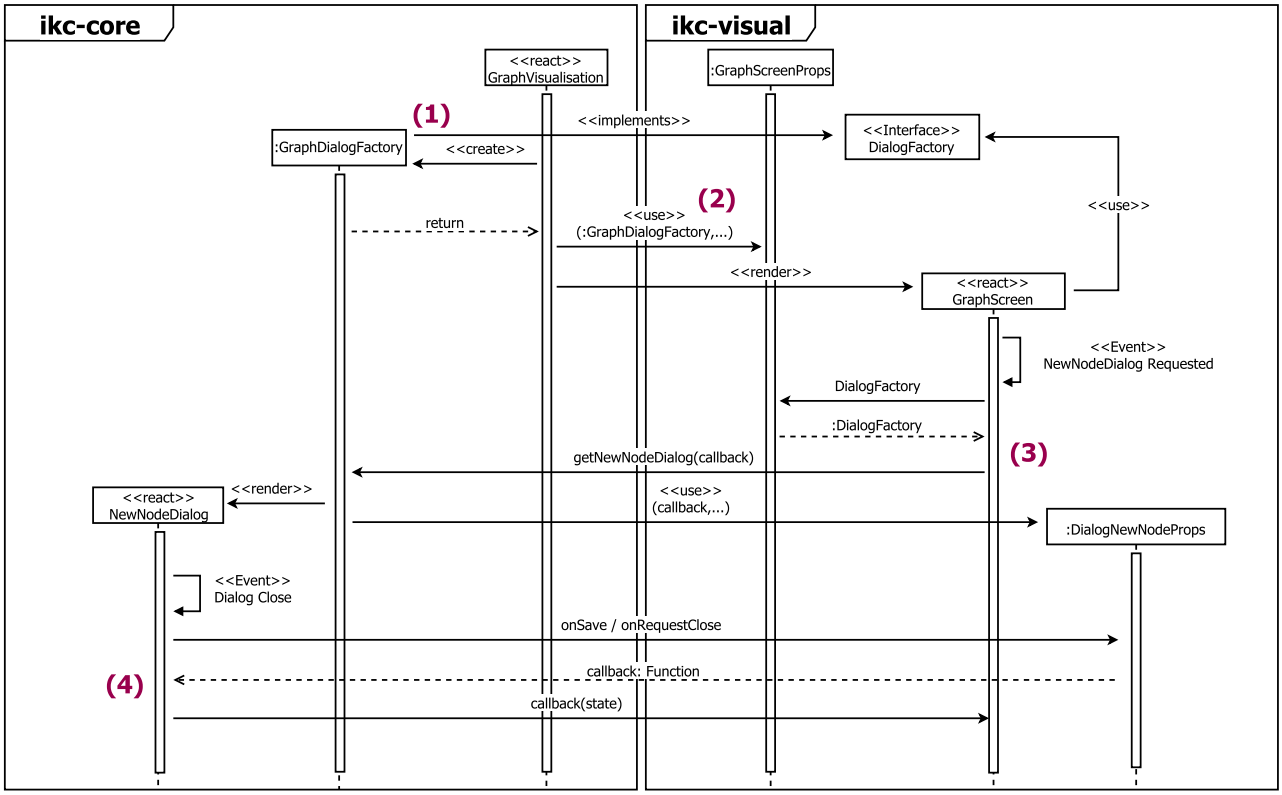


Abbildung 4.3: Interaktion mit den Factories

Weitere Implementationen

Mit Hilfe der Implantation der Schnittstellen *NodeInformationProvider*, OperationService, *IdentityService* können der Visualisierung konkrete Implementationen zu Verfügung gestellt werden (Abbildung 4.4, Abbildung 4.5, Abbildung 4.6 und Code Ausschnitt 22):

1. Das entsprechende Objekt der Implementierung wird erstellt.
2. Bei der Erstellung der Visualisierung wird das Objekt übergeben.
3. Über *GraphScreenProps* kann die Visualisierung auf das entsprechende Objekt zugreifen und mit den Methoden die benötigten

```
/** (1) ikc-core */
export class GraphDialogFactory implements
↪ DialogFactory{...}

let graphDialogFactory = new GraphDialogFactory()

/** (2) ikc-core */
<GraphScreen ... dialogFactory={graphDialogFactory} ... />

/** (3) ikc-visual */
let dialogOpen = true
let processNewNodeFromDialog = () => {...}
let closeNewNodeDialog = () => {...}
let node = GraphElementFactory.getGraphElementAsNode(...)

this.state.props.dialgoFactory.getNewNodeDialog(
    dialogOpen,
    processNewNodeFromDialog,
    closeNewNodeDialog,
    node,
    this.state.newNodeType)

/** (4) ikc-core */
this.props.onSave(this.state)
```

Listing 21: Beispiel Interaktion Factory

Informationen abrufen.

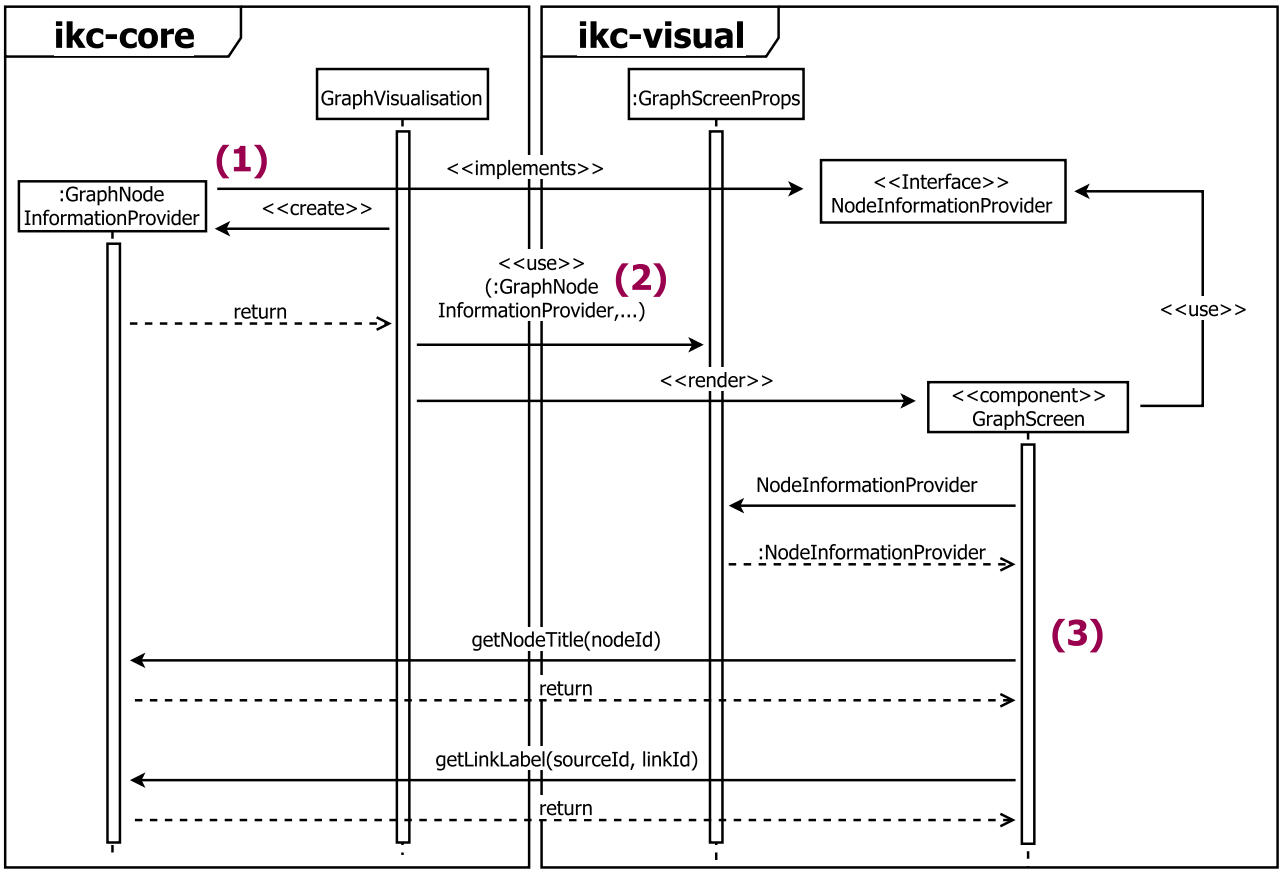


Abbildung 4.4: Interaktion mit dem NodeInformationProvider

4.4 Drag'n'Drop

Die ***Drag'n'Drop***-Funktionalität bietet sowohl auf dem mobilen als auch auf dem Desktop-Gerät die Möglichkeit, Benutzerinteraktion intuitiv und einfach zu gestalten. Wie im Unterabschnitt 3.1.3 bereits definiert, sind dies *Add Node*, *New Link*, *Link To Existing Node* und *Update Position*. Dies kann sowohl innerhalb der Visualisierung oder auch von einer externen Komponente aus geschehen. In diesem Abschnitt werden die grundsätzlichen Abläufe erläutert, welche es ermöglichen einen ***Node*** von einer externen Komponente in die Visualisierung zu ziehen. Wie die Aktionen innerhalb der Visualisierung funktionieren, wird im Unterabschnitt 4.5.1 erläutert.

Grundsätzlich wird der Vorgang des ***Drag'n'Drops*** in vier Phasen unterteilt:

```
1  /** (1) ikc-core */
2  export class GraphNodeInformationProvider implements
   ↪  NodeInformationProvider{...}
3  export class GraphOperationService implements
   ↪  OperationService{...}
4  export class ElementIdentityService implements
   ↪  IdentityService{...}
5
6  let graphNodeInformationProvider = new
   ↪  GraphNodeInformationProvider()
7  let graphOperationService = new GraphOperationService()
8  let elementIdentityService = new ElementIdentityService()
9
10 /** (2) ikc-core */
11 <GraphScreen ...
12     nodeInformationProvider={graphNodeInformationProvider}
13     operationService={graphOperationService}
14     identityService={elementIdentityService}
15     ... />
16
17 /** (3) ikc-visual */
18 let nodeInformationProvider =
   ↪  this.state.props.nodeInformationProvider
19 let operationService = this.state.props.operationService
20 let identityService = this.state.props.identityService
21
22 let nodeTitle = nodeInformationProvider.getNodeTitle('1')
23 let linkLabel = nodeInformationProvider.getLinkLabel('5')
24
25 operationService.saveView(this...)
26
27 let newNodeId = identityService.createNewNodeId()
28 let newLinkId = identityService.createNewLinkId()
29
```

Listing 22: Beispiel: Interaktion weiterer Komponenten

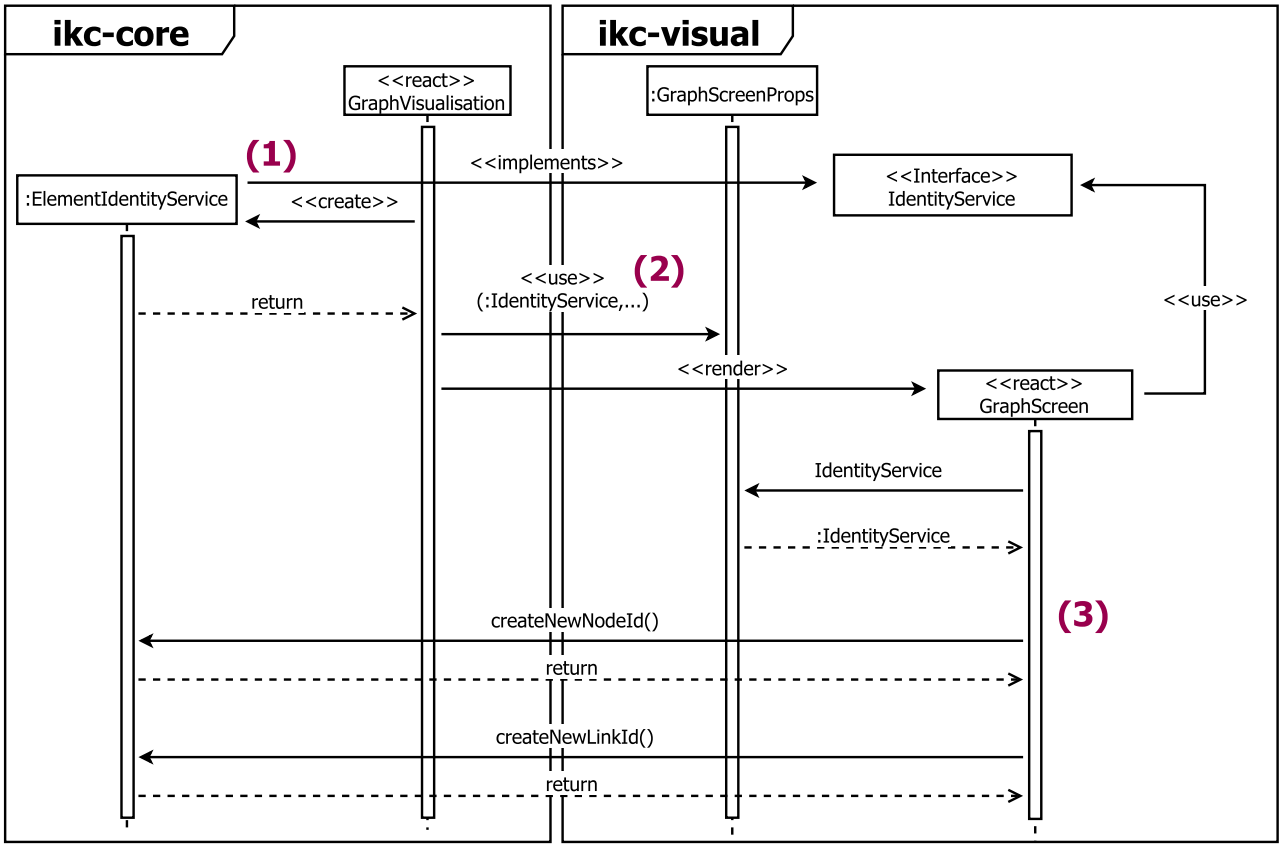


Abbildung 4.5: Interaktion mit dem IdentityService

- *Registration* - Alle Elemente, auf welche ein Drag (Nodes) oder ein Drop (Visualisierung) möglich sein soll, müssen registriert werden.
- *Drag* - Ein Drag wird mit einem Klick oder Tap auf einem Element gestartet.
- *Move* - Das Element wird auf dem Monitor bewegt.
- *Drop* - Das Element wird auf der Visualisierung losgelassen. Wenn dies an einer freien Stelle geschieht, wird der ***Node*** an dieser Stelle hinzugefügt (4.a). Geschieht dies jedoch über einem bestehenden ***Node***, wird ein neuer ***Link*** erstellt und der neue ***Node*** im Umkreis des bestehenden positioniert (4.b).

Damit diese Phasen korrekt ausgeführt werden können, ist es wichtig zu wissen, welcher ***Node*** an der Aktion beteiligt ist. Weiter müssen verschiedene ***Event***s abgefangen und verarbeitet werden. Um dies möglichst simpel und effizient zu implementieren, wird dies in eine externe ***Javascript***-Datei ausgelagert und somit vom Rest entkoppelt

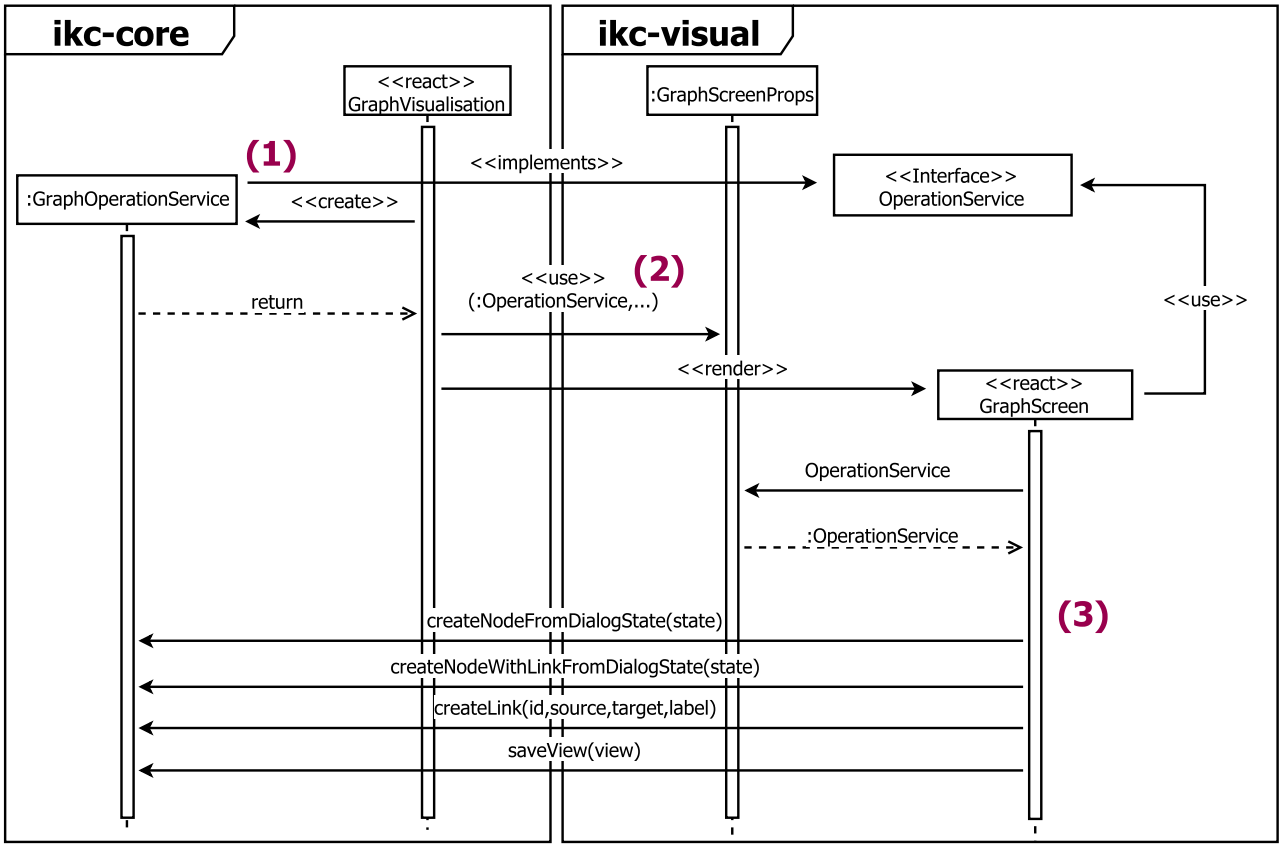


Abbildung 4.6: Interaktion mit dem OperationService

(Code Ausschnitt 23). Dieses bietet zwei zwingende und zwei optionale Methoden:

- *registerDragZone* - Mit der Registration wird das ***Drag'n'Drop*** auf dem entsprechenden ***HTML***-Element aktiviert. Dazu muss das entsprechende ***HTML***-Objekt als auch die entsprechende Node-ID übergeben werden.
- *registerDropZone* - Ebenfalls muss die Visualisierung als Drop-Zone registriert werden. Dadurch kann bei einem Drop über ihr die ***Callback***-Methode *onDrop* ausgeführt werden, welche übergeben werden muss.
- *registerCallbackForStart* - Hiermit wird jedem Element ermöglicht, bei einem Start eines ***Drag'n'Drop*** informiert zu werden.
- *registerCallbackForEnd* - Dadurch ist es möglich benachrichtigt zu werden, sobald der ***Drag'n'Drop***-Vorgang beendet ist.

Wie diese Phasen im Detail abgehandelt wird, ist in Ablaufdiagramm (Abbildung 4.8) ersichtlich. Sie sind ebenfalls in die vier Phasen des

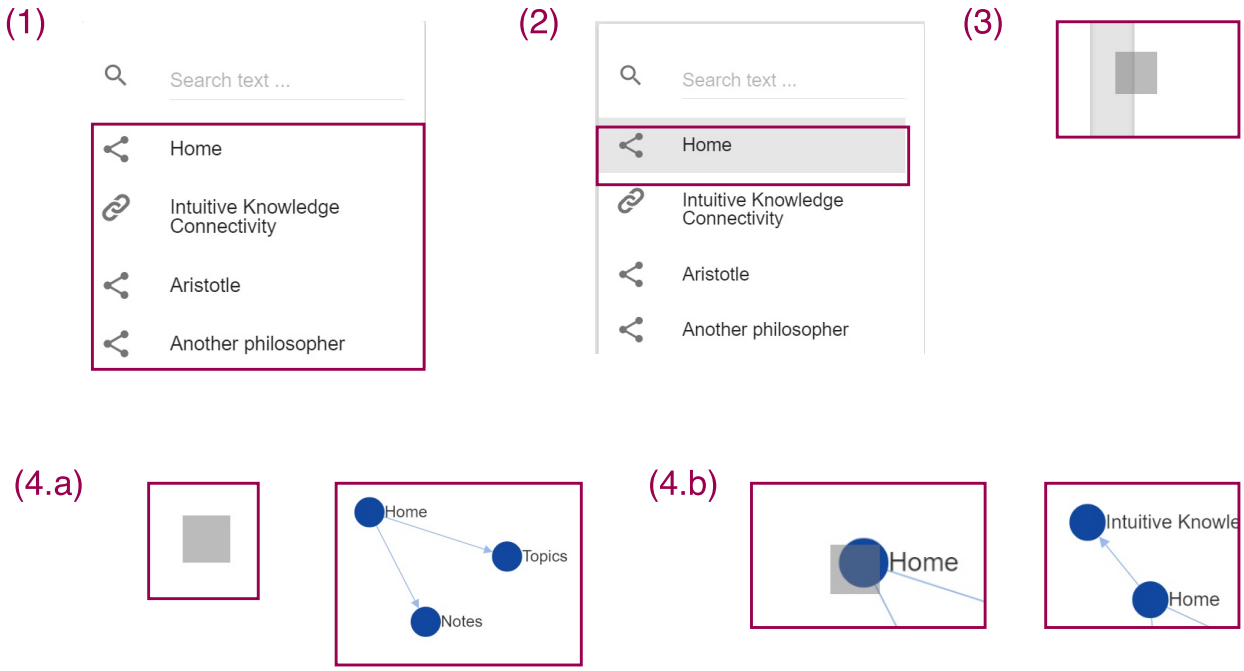


Abbildung 4.7: Übersicht ***Drag'n'Drop***

```
1 function registerDragZone(element, id){...}
2
3 function registerDropZone(onDrop){...}
4
5 function registerCallbackForStart(callback){...}
6
7 function registerCallbackForEnd(callback){...}
```

Listing 23: Drag'n'Drop Handhabung

Drag'n'Drop unterteilt.

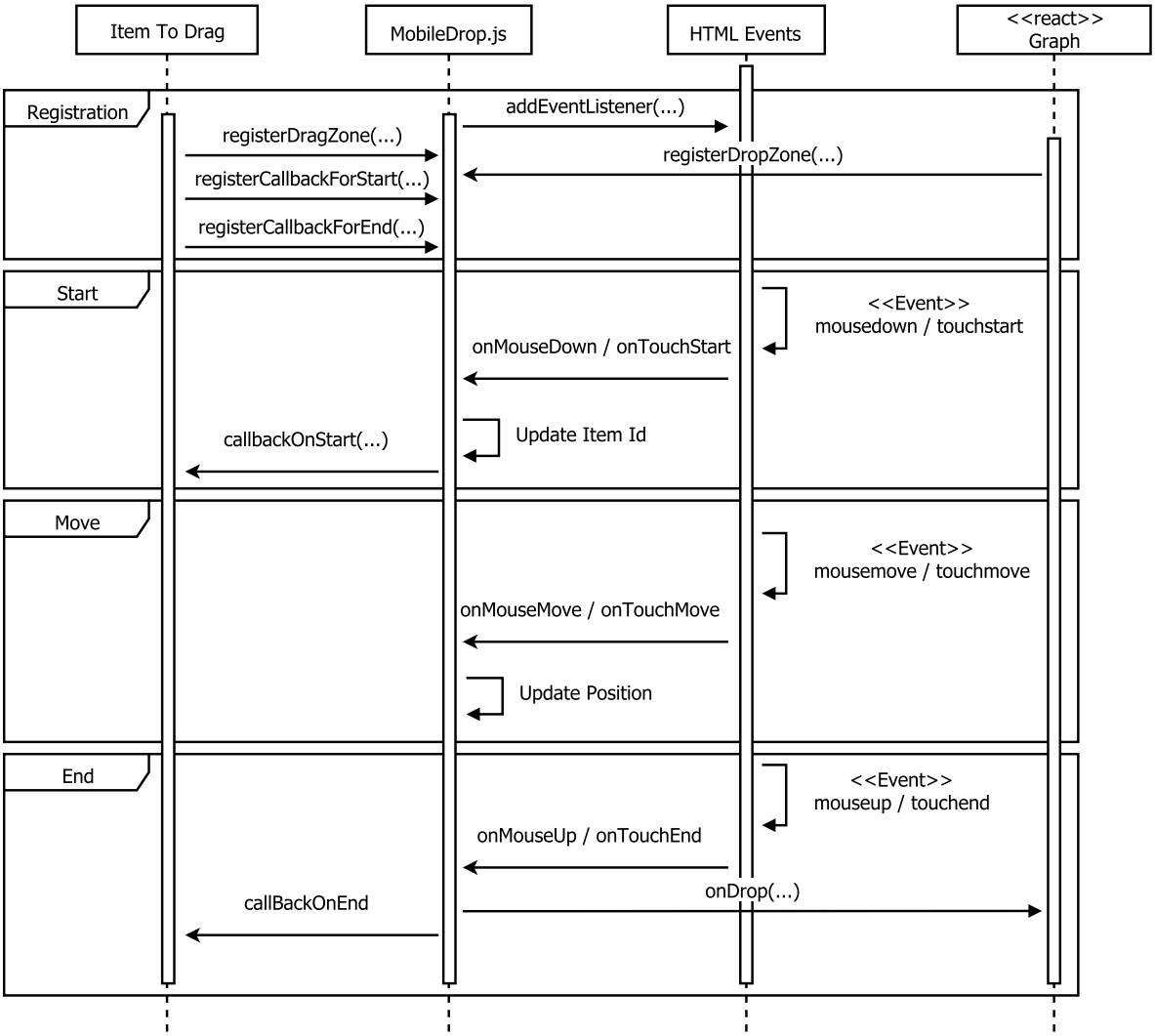


Abbildung 4.8: Ablauf ***Drag'n'Drop***

4.5 Visualisierung

Die Funktionalitäten der Visualisierung werden hauptsächlich durch vier *React*-Komponenten abgedeckt: *GraphScreen* fungiert dabei als zentrale Stelle und initialisiert alle anderen Komponenten. Insbesondere wird das entsprechende *View*-Objekt, *viewToLoad* analysiert, die darzustellenden ***Nodes*** und ***Links*** extrahiert und der *Graph*-Komponente übergeben.

Sobald innerhalb der Visualisierung ein ***Event*** aufgrund einer Benut-

zerinterkation ausgelöst wird, werden die entsprechenden ***Callback***-Methoden des *GraphScreen* aufgerufen. Darüber werden die entsprechenden Schritte ausgeführt, welche wiederum zu Aktualisierungen in den einzelnen Komponenten führen (Abbildung 4.9). Jegliche Kommunikation zwischen den Komponenten wird nach dem Prinzip des unidirektionalen Datenflusses (Unterabschnitt 4.2.3) realisiert.

Zum Beispiel soll ein ***Link*** dargestellt werden. Dazu wird im *GraphScreen* beim entsprechenden Link die Eigenschaft *visible* auf `VISIBILITY.VISIBLE` gesetzt. Durch die Aktualisierung des Zustands werden nun die ***Nodes*** und ***Links***, welche der *Graph*-Komponente übergeben werden, aktualisiert. Darin enthalten ist nun auch der neu darzustellende Link.

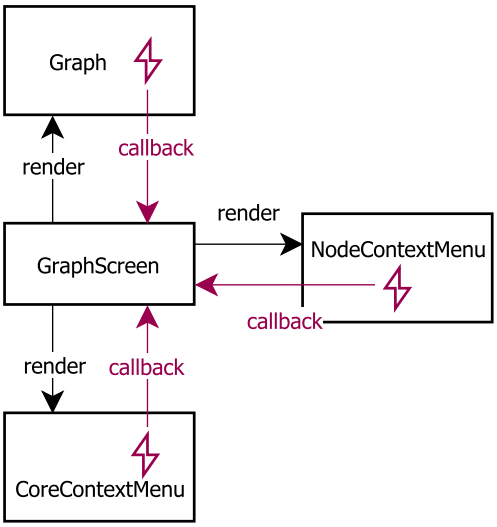


Abbildung 4.9: Komponenten der Visualisierung

4.5.1 Manipulation des Netzwerks

Die Manipulationen innerhalb des ***Netzwerks*** haben ihren Ursprung immer im *cytoscape*-Framework. Hier werden verschiedene ***Event***s registriert, um entsprechend reagieren zu können. Dies geschieht auf verschiedenen Ebenen der Visualisierung:

1. Auf der ganzen Visualisierung, zum Beispiel *Rechtsklick* an einer freien Stelle.

2. Auf mehreren ***Nodes*** der Visualisierung, zum Beispiel *durch Auswahl*.

3. Auf mehreren ***Links*** der Visualisierung, zum Beispiel durch *Auswahl*.

4. Auf einem spezifischen Element der Visualisierung, zum Beispiel *Loslassen* eines ***Nodes***.

```
1  /** (1) */
2  this.cy.on('pan', function (e: any) {...})
3
4  /** (2) */
5  this.cy.nodes().on('click', function (e: any){...})
6
7  /** (3) */
8  this.cy.edges().on('tap', function (e: any) {...})
9
10 /** (4) */
11 element.on('click', function (e: any){...})
```

Listing 24: Cytoscape Event Beispiel

Update Position/New Link (Drag'n'Drop)

Wenn die Aktion *Update Position* und *New Link* per ***Drag'n'Drop*** ausgeführt werden, startet ein teilweise ähnlicher Ablauf (Abbildung 4.10):

1. Der erste Schritt ist Teil des allgemeinen Ablaufs der Visualisierung. Darin wird die *Graph*-Komponente erstellt. Falls eine Aktion auf einem ***Node*** ausgeführt wird, wird der ***Event*** *grab* registriert.

2. Bei der Auswahl und Bewegung eines ***Nodes***, wird nun der ***Event*** *free* temporär registriert, welcher ausgeführt wird sobald der ***Node*** losgelassen wird.

3. Sobald der ***Node*** losgelassen wird, wird überprüft, ob an dieser Stelle in der Visualisierung ein anderer ***Node*** positioniert ist. Ist dies der Fall, wird ein neuer ***Link*** zwischen den zwei ***Nodes*** erstellt und dazu folgende Schritte ausgeführt:

    - Die *GraphScreen*-Komponente wird mittels einer *Callback*-Methode informiert.

- Innerhalb der *GraphScreen*-Komponente wird der neue ***Link*** erstellt und die Datenbasis durch den *OperationService* informiert.

- Nun wird der Zustand der *GraphScreen*-Komponente aktualisiert, die `render`-Methode ausgeführt und die Aktualisierungen an die *Graph*-Komponente weitergegeben. Zum Schluss werden die Änderungen angezeigt.

Wenn dies jedoch nicht zutrifft, werden folgende Schritte ausgeführt. So kann die neue Position des ***Nodes*** gespeichert werden:

- Durch die entsprechende *Callback*-Methode wird die *GraphScreen*-Komponente informiert.

- Die Position wird innerhalb der *GraphScreen*-Komponente aktualisiert und die Datenbasis durch den *OperationService* informiert.

- Ebenfalls wird der Zustand der *GraphScreen*-Komponente aktualisiert, die `render` Methode ausgeführt und die Aktualisierungen an die *Graph*-Komponente weitergegeben. Wiederum werden zum Schluss die Änderungen sichtbar.

Add Node/Link To Existing Node (Drag'n'Drop)

Im Abschnitt 4.4 wurde bereits erläutert, wie ein ***Drag'n'Drop*** eines ***Nodes*** von ausserhalb behandelt wird. Dadurch wird es ermöglicht, die Aktionen *AddNode* und *LinkToExistingNode* per *Drag'n'Drop* auszuführen. Dazu werden weitere Schritte ausgeführt, welche auf diejenigen in der Abbildung 4.8 folgen (Abbildung 4.11):

1. Auch bei diesen beiden Aktionen werden zuerst die allgemeinen Schritte abgearbeitet und die Visualisierung als *DropZone* registriert (vgl. Code Ausschnitt 23).
2. Sobald nun ein ***Node*** über der Visualisierung losgelassen wird, wird die *Graph*-Komponente informiert.
3. Anhand der Position des losgelassenen ***Nodes*** wird nun ermittelt, ob an dieser Stelle bereits ein ***Node*** in der Visualisierung dargestellt wird. Falls dies zutrifft, muss der losgelassene ***Node*** in der Visualisierung dargestellt werden und mit einem ***Link*** zum ***Node*** an der losgelassenen Stelle verbunden werden. Dafür sind folgenden Schritte notwendig:

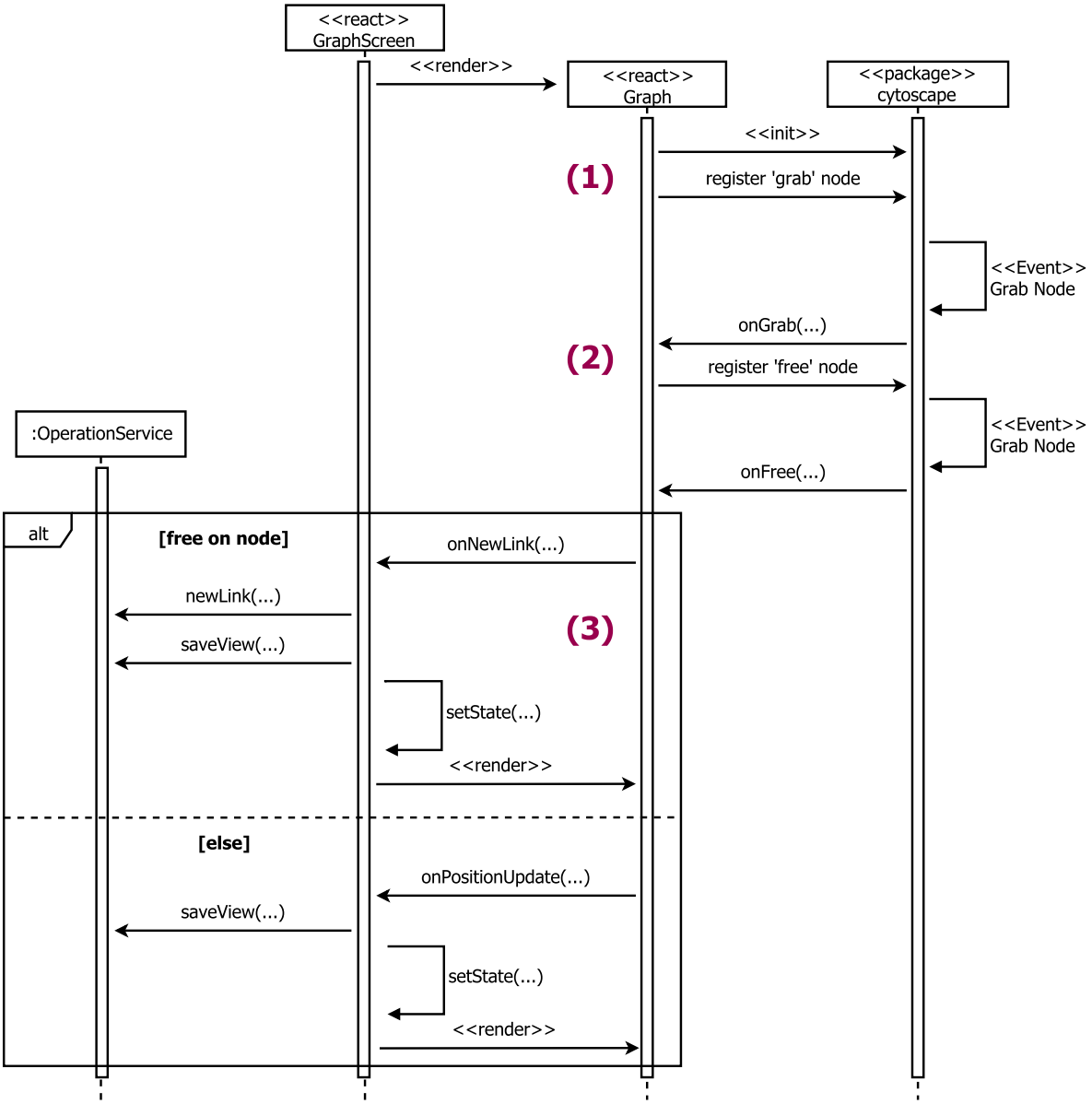


Abbildung 4.10: Ablauf Update Position / New Link

- Mithilfe der entsprechenden *Callback*-Methode wird die *GraphScreen*-Komponente informiert.

- Die Änderungen werden nun in der *GraphScreen*-omponente ausgeführt und die Datenbasis durch den *OperationService* informiert.

- Durch die Aktualisierung des Zustands der *GraphScreen*-Komponente wird die `render`-Methode ausgeführt und die Aktualisierungen an die *Graph*-Komponente weitergegeben und somit die Änderungen angezeigt.

Wenn nun der ***Node*** an einer freien Stelle losgelassen wird, soll dieser ohne einen neuen ***Link*** erstellt werden. Dazu sind folgende Schritte nötig:

- Durch die entsprechenden *Callback*-Methode wird die Information an die *GraphScreen*-Komponente weitergegeben.

- Änderungen werden nun innerhalb der *GraphScreen*-Komponente abgearbeitet und die Informationen durch den *OperationService* an die Datenbasis weitergegeben.

- Auch hier wird zum Schluss die Aktualisierung des Zustands der *GraphScreen*-Komponente ausgeführt und somit die `render`-Methode aufgerufen. Die Aktualisierungen werden an die *Graph*-Komponente weitergegeben und somit die Änderungen angezeigt.

4.5.2 Kontextmenüs

Die beiden Kontextmenüs (Abbildung 4.12a und Abbildung 4.12b) bieten eine Sammlung von *Aktionen*, welche der entsprechenden Situation angepasst sind. Beide werden entweder durch einen Rechtsklick (Desktop) oder einen langen Tap (Mobile) aufgerufen. Diese ***Event***s werden ebenfalls, wie in Abschnitt 4.5 beschrieben, innerhalb der *Graph*-Komponente beim *cytoscape*-Framework registriert. Sobald der ***Event*** auftritt werden durch die entsprechenden *Callback*-Methoden die *GraphScreen*-Komponente informiert, damit diese das entsprechende Menü öffnet.

Um die entsprechenden Suchfelder oder Dialoge aufzurufen, nutzen beide Menüs die Implementierung der Schnittstellen SearchFieldFactory oder DialogFactory (Abschnitt 4.3).

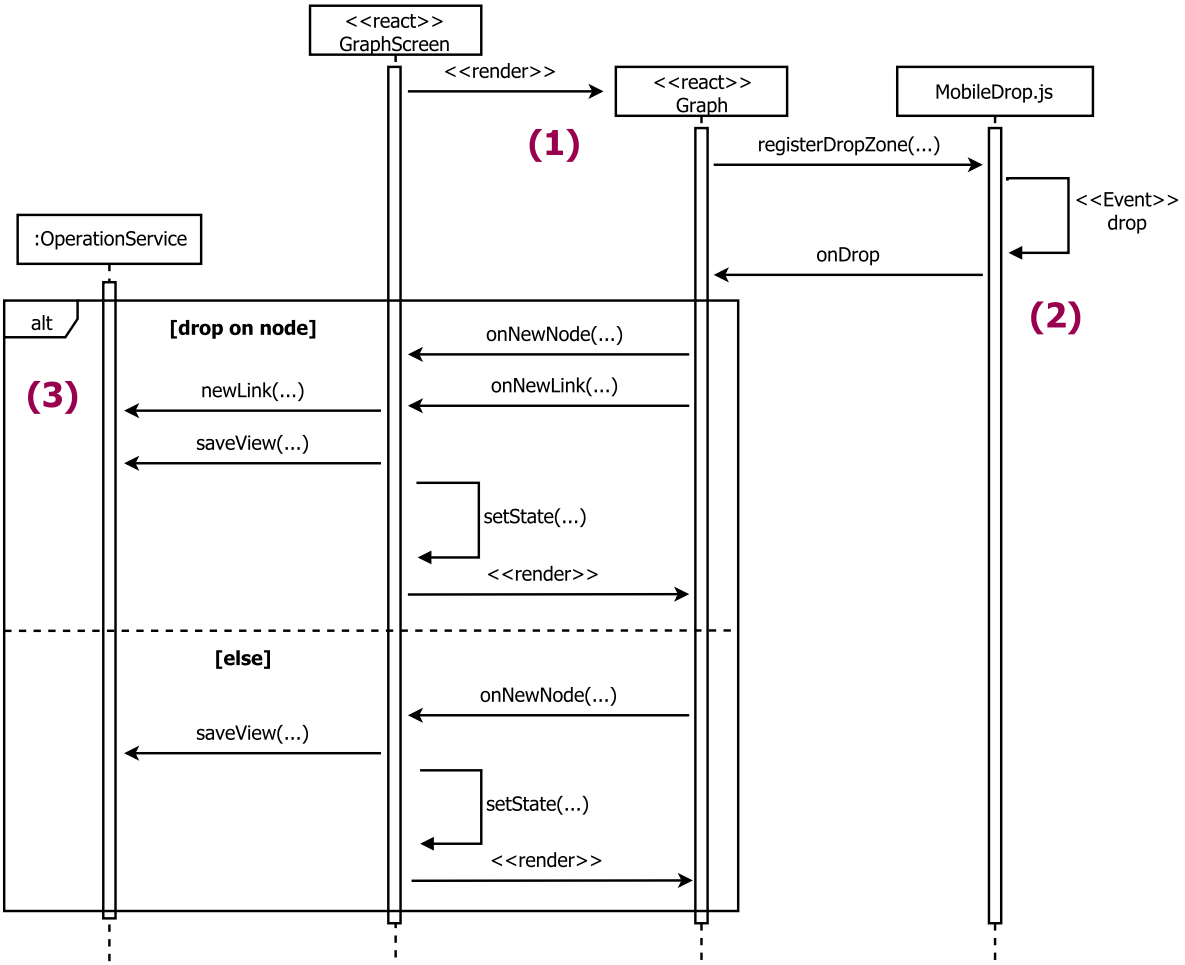


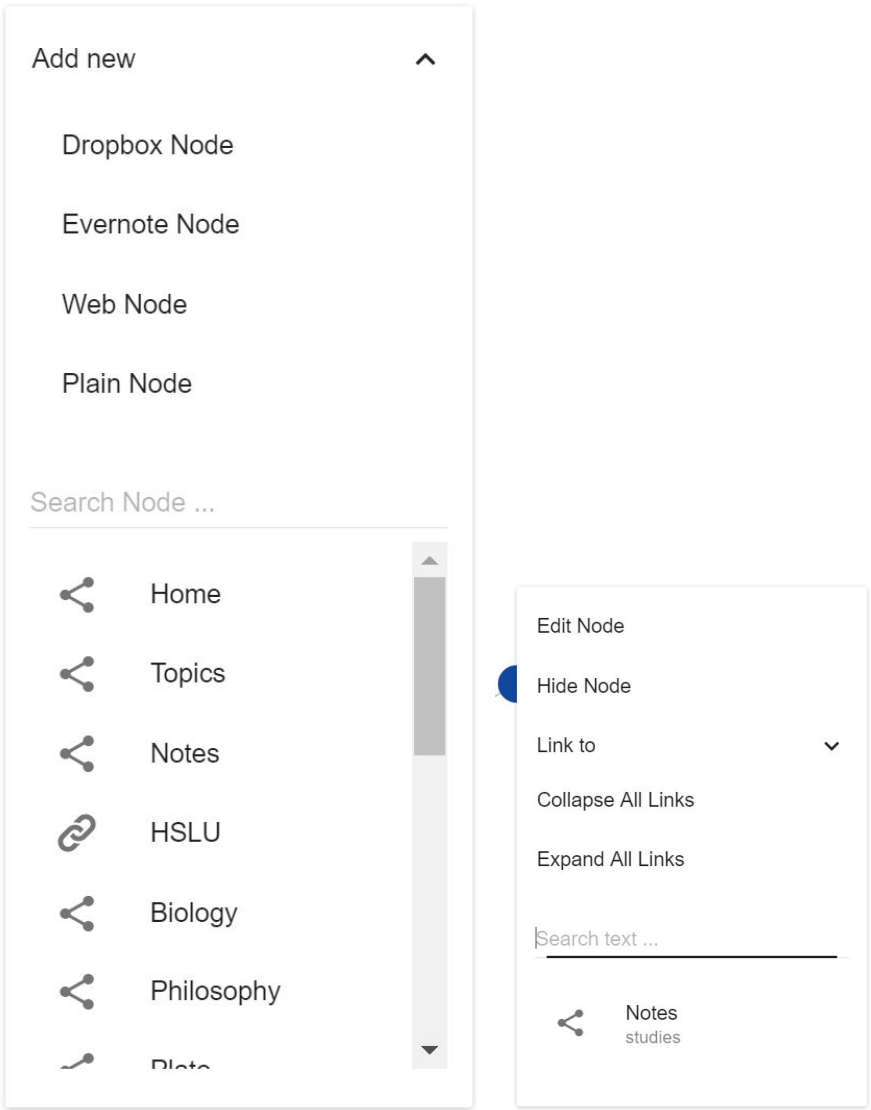
Abbildung 4.11: Ablauf Add Node / Link To Existing Node

CoreContextMenu

Im *CoreContextMenu* (Abbildung 4.12a) werden die beiden Aktionen *AddNode* und *NewNode* zur Verfügung gestellt (Unterabschnitt 3.1.3). Dabei werden durch die Implementierungen der beiden Schnittstellen SearchFieldFactory und DialogFactory das Suchfeld *GraphNodeSearchField* und der Dialog *NewNodeDialog* bezogen und eingebunden (Abbildung 4.13).

NodeContextMenu

Diverse ***Node*** spezifische Aktionen werden im *CoreContextMenu* zusammengefasst (Unterabschnitt 3.1.3). Dazu werden ebenfalls mithilfe der Implementierungen der beiden Schnittstellen SearchFieldFactory und DialogFactory das Suchfeld *GraphLinkSearchField* und die beiden



(a) CoreContextMenu (b) NodeContextMenu

Abbildung 4.12: Kontextmenüs

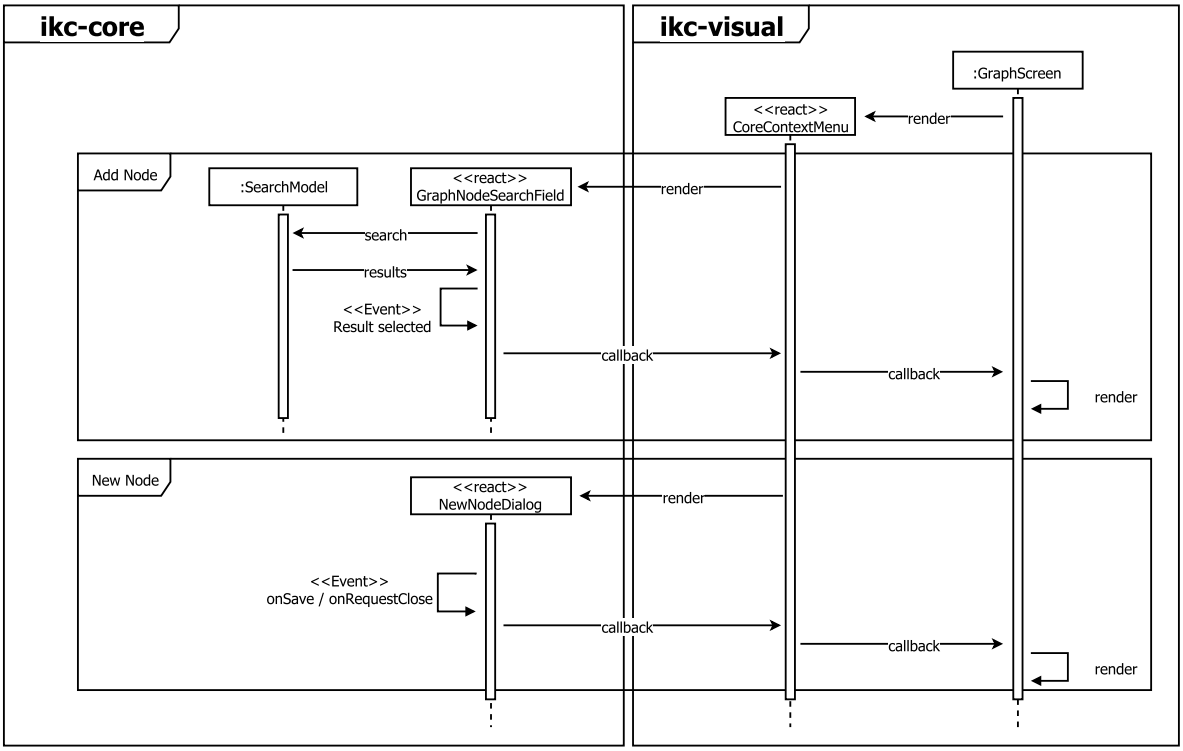


Abbildung 4.13: Ablauf CoreContextMenu

Dialoge *NewNodeToConnect* und *ExistingNodeToConnect* genutzt (Abbildung 4.14).

4.6 Integration

Mit der Integration der Visualisierung in den *ikc-core* wird die Implementation abgeschlossen. Dazu sind die nötigen Schnittstellen zu implementieren und zusätzlich weitere Anpassungen (Abbildung 4.15) notwendig:

- *GraphVisualisation* - nutzt die *GraphScreen*-Komponente und übergibt ihr alle nötigen Implementationen und Informationen.
- *GraphNodeInformationProvider* - ermöglicht der Visualisierung den Zugriff auf die Datenbasis, um Informationen abzufragen und implementiert die Schnittstelle NodeInformationProvider.
- *GraphOperationService* - Dadurch kann die Visualisierung Informationen an die Datenbasis (*NodeService*, *PropertyService* und *ViewService*) weiterleiten. Dazu wird die Schnittstelle OperationService verwendet.

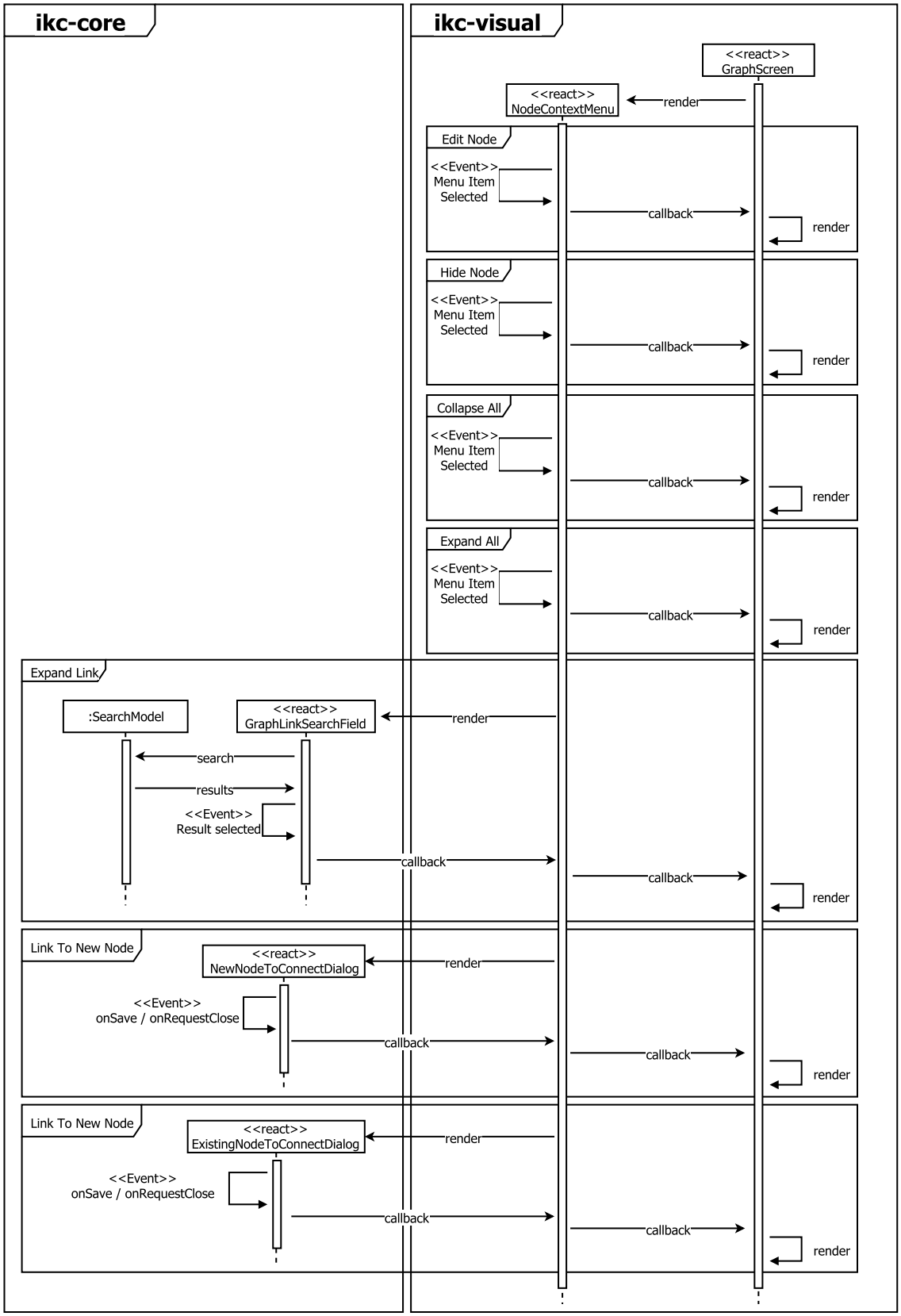


Abbildung 4.14: Ablauf NodeContextMenu

- *GraphDialogFactory* - kapselt die Erstellung der verschiedene Dialog und stellt diese zur Verfügung. Es werden die verschiedenen Methoden der Schnittstelle DialogFactory implementiert.
- *GraphSearchFieldFactory* - um die beiden Suchfelder *GraphNodeSearchField* und *GraphLinkSearchField* der Visualisierung zu Verfügung zu stellen, wird die Schnittstelle SearchFieldFactory implementiert.
- *ElementIdentityService* - damit die Visualisierung konsistente IDs für neue ***Nodes*** und ***Links*** vergeben kann, werden diese mithilfe der Schnittstelle IdentityService implementiert.
- *Dialoge* - die verschiedenen Dialoge, welche der Visualisierung durch die *GraphDialogFactory* zu Verfügung gestellt werden, nutzen die Dialog-Schnittstellen. Diese dienen als Grundlage für die ***Props***- und ***State***-Schnittstellen der *React*-Komponente.
- Um die verschiedenen ***View***s strukturiert halten zu können, werden die Klassen *ViewService* und *ViewModel* verwendet. Innerhalb des *ViewModel*s wird auch die Verbindung zur externen Persistierung auf ***Dropbox*** implementiert.
- Bisher wurde die Applikation nach folgendem Schema aufgerufen: `https://<host>/node/:node`, wobei *:node* die darzustellende Node-ID enthält, z.B. `https://localhost:8888/node/0`. Neu wird das folgende Schema verwendet: `https://<host>/:node(/:view)(/:focus)`. Dabei wurden die beiden optionalen Parameter *:view* und *:focus* hinzugefügt. *:view* enthält die entsprechende ID der View, welche darzustellen ist und *:focus* den Wert *node* oder *view* je nachdem, auf welcher Darstellung der Fokus liegt. Dies ist jedoch nur auf der Mobile-Ansicht relevant. Mit dem Aufruf `https://localhost:8888/0/0/view` wird der ***Node*** mit der ID *0* und die Sicht mit der ID *0* geladen, weiter soll der Fokus auf der Visualisierung liegen, falls die Anfrage von einem mobilen Gerät kommt.
- Der mobilen Navigationsleiste wurde ein neues Icon hinzugefügt, mit welchem zwischen der Visualisierung und der NodeDetail-Ansicht gewechselt werden kann.
- Für die Navigation in der Mobile- und der Desktop-Ansicht wurde ein neuer Punkt *View* hinzugefügt. Dadurch können neue Views erstellt, bestehende durchsucht und dargestellt werden. Weiter wurde dem Punkt *Delete* zwei Menüpunkte *DeleteView* und *DeleteNode* hinzugefügt.

- Innerhalb des *NodeService* wurden entsprechende Methoden des *ViewService* eingebaut. Somit wird sichergestellt, dass alle Änderungen der Datenbasis auch den verschiedenen Sichten weitergegeben werden. So zum Beispiel falls ein ***Link*** in der Datenbasis gelöscht wird. Diese Änderung muss auch in den Views ausgeführt werden, sodass dieser ***Link*** nicht mehr dargestellt wird.

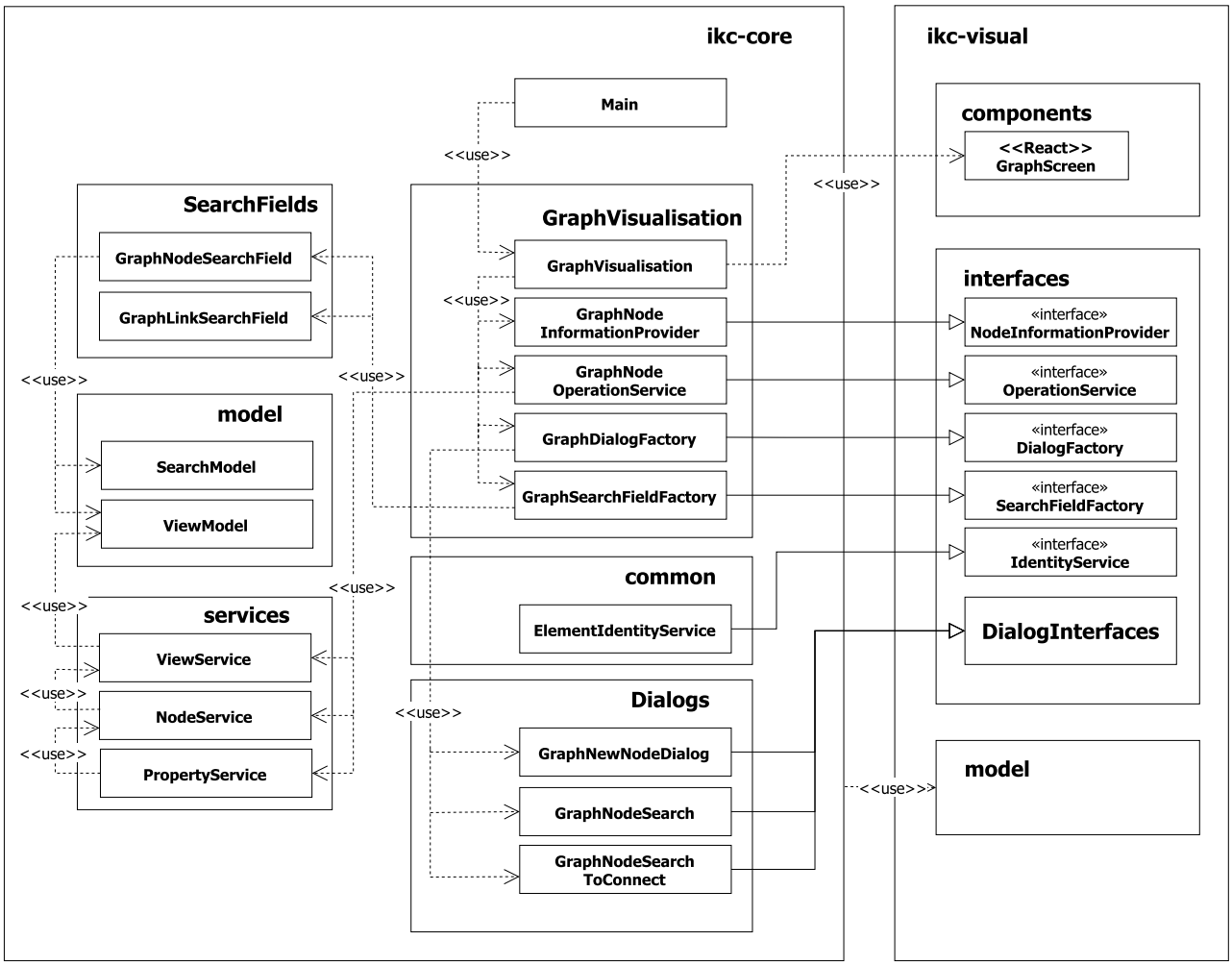


Abbildung 4.15: Intergration Visualisierung

Kapitel 5

Schlussfolgerungen

Mit dem Abschluss dieser Arbeit wird ein Prototyp einer Visualisierung zur intuitiven Interaktion mit einem Wissensnetzwerk präsentiert. Dieser Prototyp wurde losgelöst von der Hauptapplikation (***ikc-core***) entwickelt und kann als externes Software-Packet verwendet werden. Durch das Erfüllen aller Testfälle des Testkonzepts sind sämtliche Anforderungen erfüllt.

Der Prototyp ermöglicht dem Benutzer ein besseres Verständnis und einfacheren Zugang zur Datenbasis des ***ikc-cores***. Zusätzlich wurde der Funktionsumfang erweitert: Mittels den neu eingeführten ***Views*** kann ein ***Netzwerk*** weiter unterteilt oder gar kontextuell gestaltet und dargestellt.

5.1 Erkenntnisse

Während der Durchführung dieser Arbeit haben wir uns vertieft mit der Visualisierung von ***Netzwerken*** und deren Interaktion beschäftigt. Dabei haben wir folgenden Erkenntnisse gewonnen:

- Durch die Evaluationen der verschiedenen ***Framework***s zu Beginn der Arbeiten verschafften wir uns einen breiten Überblick über bestehenden Lösungen. Zum einen konnten wir ein ***Framework*** auswählen, welches unseren Anforderungen bestmöglich entspricht. Zum anderen erlangten wir dadurch einen Einblick in verschiedene Implementierungen und deren Konzept der Problemlösung Dadurch erhielten wir bereits vor der tatsächlichen Implementation ein gutes Gespür für die Thematik und deren Umsetzung, was schlussendlich zu einer guten Ressourcenplanung und Aufwandsschätzung führte.

- Das cytoscape-***Framework*** überzeugt uns für die Visualisierung von ***Netzwerken***: Es bietet eine Vielzahl von Erweiterungen, welche zusätzliche Funktionalität zur Verfügung stellen. Diese sind eigentlich für die Visualisierung von ***Netzwerken*** aus der Biologie gedacht. Welches nicht unserem Anwendungsfall entspricht. Trotzdem konnten wir verschiedene Konzepte davon aufgreifen und somit unsere Lösung zu modellieren.
- Die Darstellung von ***Netzwerken*** und deren Interaktion wird schnell unübersichtlich. Insbesondere, falls ***Links*** in beide Richtungen existieren oder dadurch Schleifen über mehrere Ebenen entstehen.
- Auch die Interaktion mit einem ***Netzwerk*** gestaltet sich schwieriger als erwartet. Beispielsweise erweist sich das Zuklappen von ***Links*** als ein Vorgang mit diversen Abhängigkeiten: Gibt es mehrere ***Links*** an einem ***Node***, wird anders reagiert als wenn nur ein ***Link*** oder keiner existiert. Somit können gleiche Aktionen je nach ***Netzwerk***-Kontext unterschiedliche Resultate ergeben.
- Die Verwendung auf mobilen Geräten mit kleinem Bildschirmen erfordert besondere Beachtung. Dabei müssen Aktionen für den Benutzer möglichst einfach erreichbar sein und gleichzeitig darf die Applikation nicht überladen wirken. Die Bedienung mit den Fingern hat andere Anforderungen als eine Bedienung mit der Maus und Tastatur.
- Die Interaktion mit einer Benutzeroberfläche kann bereits vor der Implementierung konzeptionell genau geplant werden. Die Missachtung von kleinen Details kann unter Umständen später zu Unklarheiten und erheblichem Zusatzaufwand führen.

5.2 Lessons learned

Basierend auf den Erkenntnissen und der Reflexion unserer Arbeit, sind die folgenden Punkte, die wichtigsten Schlüsse, welche wir aus dem Modul mitnehmen:

- Die Anforderungen an die Benutzeroberfläche und deren Bedienung müssen möglichst früh genau festgelegt werden. Obwohl wir dies zum Teil gemacht haben, tauchten im Laufe der Entwicklung diverse Unklarheiten auf. Mit detaillierteren Anforderungen und genaueren Modellen, hätten diese vermieden werden können.

- Durch das punktuelle Ein- und Ausblenden der Beschriftungen von ***Links*** kann die Übersichtlichkeit in grossen Netzwerken zusätzlich erhöht werden.
- Die vom ***ikc-core*** losgelöste Entwicklung erzwang eine komplett eigenständige Entwicklung gleich von Beginn an. Dadurch war eine lose Kopplung und gleichzeitige hohe Kohäsion stets gewährleistet.
- Aufgrund der internationalen Projektorganisation musste den Methoden und den Mittel für eine Kollaboration besondere Beachtung geschenkt werden. Mit den angegebenen Hilfsmitteln funktionierte dies ausgezeichnet. Trotz den gegebenen Umständen war eine enge Zusammenarbeit und ein ständiger Austausch stets gegeben. Sehr hilfreich waren dabei auch Bildschirmaufzeichnungen. Die grösste Herausforderung stellte sicherlich die zeitlichen Koordinaten dar. Besprechungen wurden darum möglichst regelmässig zu fixen Zeiten abgehalten.

  Die Auslastung konnte stets gut der aktuellen Situation angepasst werden. So konnten sich Andreas Waldis im Dezember und Patrick Siegfried im Januar praktisch ausschliesslich den Prüfungen widmen. Auch konnte an Teile der Arbeit gut unabhängig gearbeitet werden, sodass der Arbeits-Fortschritt trotz zeitweiser intensiver Auslastung nie ausblieb. Eine Erleichterung war die Erweiterung des Zeitrahmens seitens der HSLU. Dadurch war es einfacher möglich die Arbeit trotz den unterschiedlichen Semesterterminen erfolgreich abzuschliessen.

5.3 Ausblick

Der präsentierte Prototyp erfüllt die Anforderungen und ist in den ***ikc-core*** integriert. Jedoch gibt es verschiedene mögliche Optimierungen:

- Mit Hilfe der Erweiterung *cytoscape.js-undo-redo* können Operationen rückgängig gemacht werden. Dies könnte zum Beispiel eine nutzliche Funktion sein, um versehentlich gelöschte Elemente wiederherzustellen. (iVis-at Bilkent, 2017b).
- Beim Hinzufügen eines neuen ***Nodes*** wird dieser automatisch zufällig unter dem Eltern-***Node*** angezeigt. Dies kann problematisch werden, falls exakt an dieser Stelle bereits ein ***Node*** angezeigt wird. Dies könnte verhindert werden, indem die Positionen aller bereits angezeigten Nodes berücksichtigt würde.

- Durch die Verwendung von verschiedenen Farben, Formen und Grössen wäre es möglich ***Nodes*** unterschiedlicher grafisch zu differenzieren, und so unterschiedliche ***Node***-Typen sichtbar hervorzuheben. Die Erweiterung *cytoscape.js-supportimages* würde eine Einbindung von Piktogrammen ermöglichen. (jhonatandarosa, 2017).
- Die Erweiterung *js-cytoscape-navigator* zeigt eine Übersicht über das dargestellte Netzwerk an. Der Vorteil ist vor allem die erhöhte Übersicht in grossen ***Netzwerken***.
- Die Erweiterung *cytoscape.js-clipboard* bietet 'Copy-Paste'-Funktionalität. Damit könnten Teile einer ***View*** wiederverwendet werden. (iVis-at Bilkent, 2017a).
- Eine denkbare Erweiterung ist eine automatische Generierung von ***Views***. Diese würde dem Benutzer eine Übersicht über sein Wissen bieten. Es ist vorstellbar, dass er bestimmte Parameter, beispielsweise die Tiefe des darzustellenden Netzwerks von einem gewählten Ausganspunkt aus, beeinflussen kann.
- cytoscape hat seine Ursprünge in der ***Netzwerk***-Theorie. Darum bietet es neben der Darstellung auch diverse Algorithmen an. Dazu gehören beispielsweise bekannte Algorithmen *Dijkstra* oder auch *A**. (Consoritum, 2017)
- Ab einer gewissen ***Node***-Titellänge wird die Darstellung ungünstig. Der Titel ragt über die jeweilige Komponente. Dies könnte einfach mit ***CSS*** oder ***Javascript*** gelöst werden.

Literaturverzeichnis

A book apart: Responsive web design - ethan marcotte. (2010). Book Apart.

Bradner, S. (1997, 03). *Key words for use in rfcs to indicate requirement levels*. http://www.ietf.org/rfc/rfc2119.txt.

call-em all. (2016). Zugriff am 2017-1-23 auf http://www.material-ui.com

Consoritum, C. (2017). Zugriff am 2017-1-29 auf http://js.cytoscape.org/

Dropbox. (2017). Zugriff am 2017-1-29 auf http://www.searchstorage.de/definition/Dropbox

Duden. (2017). Zugriff am 2017-1-29 auf http://www.duden.de/rechtschreibung/Cloud_Computing

Eichinger, A. (2017). Zugriff am 2017-1-29 auf http://www.uni-regensburg.de/Fakultaeten/phil_Fak_II/Psychologie/Doktoranden/absolventen/eichinger_armin/u-definition.html

Evernote. (2017). Zugriff am 2017-1-29 auf http://searchmobilecomputing.techtarget.com/definition/Evernote

Facebook. (2017). Zugriff am 2017-1-23 auf https://facebook.github.io/react/

Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O. & Bader, G. D. (2016). *Cytoscape.js: a graph theory library for visualisation and analysis*. Zugriff am 2016-11-9 auf http://bioinformatics.oxfordjournals.org/content/32/2/309

Google. (2016). Zugriff am 2017-1-23 auf https://material.io/guidelines/material-design/introduction.html

Gründerszene. (2017). Zugriff am 2017-1-29 auf http://www.gruenderszene.de/lexikon/begriffe/user-experience

iVis-at Bilkent. (2017a). Zugriff am 2017-1-29 auf https://github.com/iVis-at-Bilkent/cytoscape.js-clipboard

iVis-at Bilkent. (2017b). Zugriff am 2017-1-29 auf https://github.com/iVis-at-Bilkent/cytoscape.js-undo-redo

jhonatandarosa. (2017). Zugriff am 2017-1-29 auf https://github.com/jhonatandarosa/cytoscape.js-supportimages

jsplump. (2017). Zugriff am 2017-1-25 auf https://jsplumbtoolkit.com/

Kaufmann, M., Waldis, A., Siegfried, P., Wilke, G., Portmann, E. & Hemmje, M. (2016, 10). *Intuitive knowledge connectivity*. http://link.springer.com/chapter/10.1007%2F978-3-319-47650-6_27.

mxGraph. (2017). Zugriff am 2017-1-25 auf https://www.jgraph.com/

Poppe, M. (2016). Zugriff am 2016-11-9 auf https://github.com/maurizzzio/greuler

Rappid. (2016). Zugriff am 2016-12-12 auf http://resources.jointjs.com/docs/jointjs/v1.0/joint.html

Scala-Lang. (2017). Zugriff am 2017-1-25 auf http://docs.scala-lang.org/tutorials/tour/case-classes.html

Typescript. (2017). Zugriff am 2017-1-22 auf https://www.typescriptlang.org

webpack. (2016). Zugriff am 2017-1-23 auf https://webpack.github.io/docs/motivation.html

Wheeler, K. (2016). Zugriff am 2017-1-23 auf https://scotch.io/tutorials/learning-react-getting-started-and-concepts

Glossar

Array Datenstruktur, welche eine Sammlung von Elementen enthält. 37

Callback Methodenreferenz als Parameter. So kann eine einseitige Kopplung realisiert werden. 29, 34, 36, 37, 55, 57, 63, 65

Canvas ***HTML***-Element, welches zum Zeichnen benutzt werden kann. 48

cloudbasiert Cloud-Computing: Nutzung von IT-Infrastrukturen und -Dienstleistungen, die nicht vor Ort auf lokalen Rechnern vorgehalten, sondern als Dienst gemietet werden und auf die über ein Netzwerk (z.B. das Internet) zugegriffen wird Duden (2017). 3

CSS Cascading Stylesheets, ist eine Computersprache für die Gestaltung von ***HTML***-Elementen. 48, 80

Drag'n'Drop Methode zur Bedienung einer Benutzeroberfläche, wobei ein Element mit der Maus oder dem Finger 'gepackt' und anschliessend 'losgelassen' wird. 10, 15, 16, 17, 21, 24, 30, 44, 60, 63, 62, 63, 67, 68, 86, 91, 93

Dropbox Cloud-Service zum Austausch von Daten Dropbox (2017). 3, 15, 75

Event ***HTML*** oder ***Javascript***-Ereignis, welche beispielsweise durch eine Benutzerinteraktion ausgelöst wird. Diese Ereignisse können im ***Javascript*** abgefangen und so darauf entsprechend reagiert werden. 5, 30, 55, 62, 65, 66, 67, 70

Evernote Web-basierte Applikation zur Erfassung, Speicherung und Synchronisation von Text, Bildern oder Videos Evernote (2017). 3, 15

Framework Programmatische Erweiterung oder Zusatzbibliothek. II, 15, 16, 26, 43, 44, 45, 46, 48, 77

HTML Hypertext Markup Language, welche die Struktur einer Webseite bildet. 48, 53, 63, 83

IKC Forschungsprojekt der Hochschule Luzern - Informatik, welches sich mit plattformübergreifenden Wissennetzwerken beschäftigt. 5, 14

ikc-core Bestehender Software-Prototyp aus dem IKC-Projekt. II, 3, 5, 6, 8, 10, 12, 10, 12, 15, 16, 19, 24, 25, 26, 28, 29, 30, 32, 34, 37, 51, 57, 77, 79, 85, 86, 93

Intuitive Knowledge Connectivity Forschungsprojekt der Hochschule Luzern - Informatik, welches sich mit plattformübergreifenden Wissennetzwerken beschäftigt. 1, 3

Javascript Skriptsprache für Webprogrammierung. 62, 80, 83

Link verbindet zwei ***Nodes*** in einem Netzwerk. 10, 17, 19, 20, 21, 22, 25, 29, 32, 36, 38, 41, 42, 43, 47, 62, 65, 66, 67, 68, 70, 75, 78, 84

mobile first Der Prototyp wird in erster Linie für die Darstellung auf mobilen Endgeräten (Tables und Smartphones) entwickelt. 10

MoSCoW-System Technik zur Priorisierung von Anforderungen Bradner (1997). 10

Netzwerk Netzwerk als Sammlung von ***Nodes*** und ***Links***. II, 10, 15, 19, 20, 21, 24, 28, 29, 42, 46, 47, 66, 77, 78, 80

Node Element innerhalb eines Netzwerk.. 10, 17, 19, 20, 21, 22, 24, 25, 29, 32, 34, 36, 38, 39, 40, 41, 42, 43, 44, 47, 60, 62, 65, 66, 67, 68, 70, 71, 75, 78, 79, 80, 84, 91, 93

PAWI Informatik-Projekt der Studierenden an der Hochschule Luzern.. 3, 5, 15

Props React-spezifische Eigenschaften von Komponenten (vgl. React: State-Schnittstelle. 75

Responsive Design Webseiten sollen nicht nur im Layout / Design nicht nur flexibler werden, sondern sich dem Medium anpassen (*A Book Apart: Responsive web design - Ethan Marcotte*, 2010, S.8). 6, 15

Scala JVM-basierte Programmiersprachen mit modernen Sprachkonstrukten. 38

State React-spezifische Zustände von Komponenten (vgl. React: Props-Schnittstelle. 75

Tags Tags welche im ***ikc-core*** benutzt werden können, um Nodes zu gruppieren. 19

Toolbox Werkzeugplalette, welche beispielsweise Zusatzfunktionalitäten bietet. 44

Usability Usability eines Produktes ist das Ausmaß, in dem es von einem bestimmten Benutzer verwendet werden kann, um bestimmte Ziele in einem bestimmten Kontext **effektiv**, **effizient** und **zufriedenstellend** zu erreichen Eichinger (2017). 5, 85

User Interface Das User Interface ist für den Benutzer die (grafische) Schnittstelle zum Prototypen. 5

User Experience User Experience erweitert das Konzept der ***Usability*** um ästhetische und emotionale Faktoren Gründerszene (2017). 5

View Benutzerbasierte Visualisierung eines (Teil-)Netzwerks. 10, 17, 32, 42, 75, 77, 80, 91, 93

Abbildungsverzeichnis

2.1 Projektstrukturplan 7
2.2 Rahmenplan 9

3.1 Skizzen: Benutzeroberfläche 20
3.2 Skizzen: Benutzeroberfläche 21
3.3 Skizze: NodeContextMenu 21
3.4 Skizzen: Aktionen 24
3.5 Skizzen: Aktionen 24
3.6 Grenze ***ikc-core*** - visual 25
3.7 3-spaltige Oberfläche 26
3.8 Komponentendiagramm 26
3.9 Klassendiagramm 27
3.10 Integration ***ikc-core*** 28
3.11 Beispiel Cytoscape 47
3.12 Beispiel Greuler 48
3.13 Frameworks 49

4.1 Überblick Implementation 52
4.2 Unidirektionaler Datenfluss 56
4.3 Interaktion mit den Factories 58
4.4 Interaktion mit dem NodeInformationProvider 60
4.5 Interaktion mit dem IdentityService 62
4.6 Interaktion mit dem OperationService 63
4.7 Übersicht ***Drag'n'Drop*** 64
4.8 Ablauf ***Drag'n'Drop*** 65
4.9 Komponenten der Visualisierung 66
4.10 Ablauf Update Position / New Link 69
4.11 Ablauf Add Node / Link To Existing Node 71
4.12 Kontextmenüs 72
4.13 Ablauf CoreContextMenu 73

4.14 Ablauf NodeContextMenu 74
4.15 Intergration Visualisierung 76

Code

1 GraphScreen-Komponente 30
2 GraphScreen-Beispiel 31
3 NodeInformationProvider-Interface 32
4 OperationService-Interface 33
5 IdentityService-Interface 33
6 DialogFactory-Interface 35
7 DialogNewNode-Schnittstellen 36
8 DialogNewNodeToConnect-Interfaces 37
9 DialogSearchNodeToConnect-Interfaces 38
10 SearchFieldFactory-Schnittstelle 39
11 GraphElement-Schnittstelle 39
12 GraphElementData-Interface 40
13 GraphNodeElement-Klasse 40
14 GraphNodeData-Klasse 41
15 GraphLinkElement-Klasse 41
16 GraphLinkData-Klasse 42
17 GraphPosition-Klasse 42
18 View-Klasse 43
19 VISIBILITY-Enumerator 44
20 Beispiel React Komponente 54
21 Beispiel Interaktion Factory 59
22 Beispiel: Interaktion weiterer Komponenten 61
23 Drag'n'Drop Handhabung 64
24 Cytoscape Event Beispiel 67

Anhang A

Appendix

A.1 Testfälle

ID	Ablauf	Erfüllt
T1	Das (allgemeine) Kontextmenü öffnen und im Untermenü *Add New* den entsprechenden Typ auswählen. Danach öffnet sich ein Dialog, wo der neue ***Node*** beschrieben werden kann. Nach dem Speichern muss er an der Stelle, wo das Kontextmenü aufgerufen worden ist, angezeigt werden.	Ja
T2	Das ***Node***-Kontextmenü öffnen und im Untermenü *Link To* den entsprechenden Typ auswählen. Danach öffnet sich ein Dialog, wo der neue ***Node*** beschrieben werden kann. Nach dem Speichern muss der Node, bei welchem das Kontextmenü aufgerufen wurde, mit einem Link zum neuen Node verbunden sein und in der Umgebung darstellt werden.	Ja
T3	Das (allgemeine) Kontextmenü öffnen und im unteren Bereich den darzustellenden ***Node*** suchen und auswählen. Danach muss er an der Stelle, wo das Kontextmenü aufgerufen worden ist, zusammen mit seinen Kind-***Nodes*** und den verbindenden Links ersichtlich sein.	Ja

T4	Das ***Node***-Kontextmenü öffnen und im Untermenü *Link To Existing Node* auswählen. Danach öffnet sich ein Dialog, wo der bestehende ***Node*** gesucht und ausgewählt werden kann. Nach dem Speichern muss dieser mit dem Node, auf dem das Kontextmenü aufgerufen wurde, mit einem Link verbunden sein und in der Umgebung dargestellt werden.	Ja
T5	Aus der Suche einen ***Node*** per ***Drag'n'Drop*** in die Visualisierung ziehen und an einer freien Stelle loslassen. An dieser Stelle muss nun dieser ***Node*** dargestellt werden. Für die mobile Ansicht muss zuerst die Suche über das entsprechende Icon geöffnet werden. Sobald der ***Drag'n'Drop*** Prozess beginnt, verschwindet das Suchfeld, um den ***Node*** optimal positionieren zu können, nach dem Loslassen wird es wieder dargestellt.	Ja
T6	Aus der Suche einen ***Node*** per ***Drag'n'Drop*** in die Visualisierung ziehen und über einem bereits dargestellten ***Node*** loslassen. Nun muss dieser ***Node*** mit dem ausgewählten ***Node*** per Link verbunden werden. Für die mobile Ansicht muss zuerst die Suche über das entsprechende Icon geöffnet werden. Sobald der ***Drag'n'Drop***-Prozess beginnt, verschwindet das Suchfeld, um den ***Node*** optimal positionieren zu können, nach dem Loslassen wird es wieder dargestellt.	Ja
T7	Das ***Node***-Kontextmenü öffnen und mit *Hide Node* den entsprechenden ***Node*** ausblenden. Dabei müssen alle ein- und ausgehenden Links ebenfalls ausgeblendet werden.	Ja
T8	Das ***Node***-Kontextmenü öffnen und mit *CollapseAllLinks* alle ausgehenden Links ausblenden. Falls der Ziel-***Node*** des jeweiligen ***Nodes*** keinen anderen ein- oder ausgehenden Link besitzt, muss dieser auch ausgeblendet werden.	Ja
T9	Das ***Node***-Kontextmenü öffnen und mit *ExpandAllLinks* alle ausgehenden Links einblenden. Falls der Ziel-***Node*** nicht bereits eingeblendet ist, muss dieser zusammen mit dem Link eingeblendet werden.	Ja

T10	Das ***Node***-Kontextmenü öffnen und im unterem Bereich einen ausgehenden Link auswählen, dieser muss anschliessend zusammen mit dem Ziel ***Node***, falls dieser noch nicht dargestellt ist, dargestellt werden.	Ja
T11	Die auszublendenden Links in der Visualisierung auswählen, diese werden pink gefärbt. Danach in der Toolbar mit *COLLAPSE* ausblenden.	Ja
T12	In der Navigation im Menu *View*, *New View* auswählen. Die Visualisierung wird aktualisiert mit einer leeren ***View***.	Ja
T13	Eine neue ***View*** erstellen einen beliebigen ***Node*** darstellen, danach den Browser schliessen und neu öffnen. Nun muss die zuvor erstellte ***View*** ebenfalls verfügbar und der eben erstelle ***Node*** sollte sichtbar sein.	Ja
T14	Titel eines ***Nodes*** in der Datenbasis aktualisieren. Dieser muss auch in der Visualisierung aktualisiert sein.	Ja
T15	Einen ***Node*** in der Datenbasis löschen. Dieser muss auch aus allen ***View*** gelöscht sein.	Ja
T16	Einen Link aus der Datenbasis löschen. Dieser darf nicht mehr in den ***View*** dargestellt werden.	Ja
T17	Einen Link in der Datenbasis erstellen. Dieser darf nicht dargestellt werden, muss aber bei dem entsprechenden ***Node*** zur Auswahl stehen für die Darstellung.	Ja
T18	Einen bestehenden ***Node*** mit mehr als sieben ausgehenden ***Nodes*** darstellen. Es dürfen dabei nicht mehr als sieben Links dargestellt werden.	Ja
T19	In der Toolbar mit *SHOW LINK LABEL* bzw. *HIDE LINK LABEL* die Labels der Links ein oder ausblenden.	Ja

Tabelle A.1: Testfälle Beschreibung

A.2 User-Stories

ID	Description
S1	Die Schnittstellen und die Model-Klassen für die Implementierung definieren.

S2	Das Projektteam erhält einen Überblick über den aktuellen Stand der Technik. Es soll eine Auswahl von vier möglichen Frameworks getroffen werden.
S3	Detail Beurteilung zweier Frameworks anhand des Standard-Szenarios.
S4	Detail Beurteilung zweier Frameworks anhand des Standard-Szenarios.
S5	Definition des Standard-Szenarios zur genaueren Beurteilung der vier möglichen Frameworks.
S6	Mock-up aller Benutzeroberflächen erstellen
S7	Anbindung des Interface-Paket (*ikc-visual*) via NPM Setup. Typescript- und React-Setup des Projekts im Gitlab
S8	Der Benutzer kann einen ***Node*** z.B. aus der Suche von ausserhalb in die Visualisierung ziehen. Wird der ***Node*** an einer freien Position losgelassen, wird dieser der Visualisierung an dieser Stelle hinzugefügt. Wird er jedoch über einem bestehenden ***Node*** losgelassen, wird ein neuer Link zu diesem erstellt und der neue ***Node*** in seiner Umgebung platziert. Dies soll sowohl im mobilen als auch auf im Desktop-Umfeld möglich sein. Der entsprechende ***Node*** wird immer zusammen mit all seinen Kindern angezeigt. Sind es mehr als sieben, werden nur sieben angezeigt und der Rest über das Kontext-Menü zugänglich gemacht.
S9	Einen ***Node*** in der Visualisierung kann per ***Drag'n'Drop*** positioniert werden. Diese Position muss gespeichert werden.
S10	Werden zwei ***Nodes*** in der Visualisierung aufeinander gezogen, sollen zwei beschriftete Links entstehen, jeweils in eine Richtung.
S11	Mittels eines Kontext-Menüd kann der Benutzer neue ***Nodes*** erstellen und die Visualisierung hinzufügen oder einen bestehenden hinzufügen. Dies geschieht an der Stelle, wo das Menü aufgerufen wurde. Das Menü öffnet sich durch einen Rechtsklick oder einen langen Tap. Das Suchfeld wird vom ***ikc-core*** geliefert werden.

S12	Der Benutzer kann durch einen Rechtklick auf einen ***Node*** oder einen langen Tap das ***Node*** Kontext-Menü aufrufen. Dort soll er die folgenden Möglichkeiten haben: *Edit Node*, *Hide Node LinkToExistingNode*, *LinkToNewNode* (Link zu einem bestehenden ***Node*** erstellen oder einen neuen ***Node*** erstellen und dann verlinken, der entsprechende ***Node*** wird in der Umgebung angezeigt), *CollapseAllLinks* (alle ausgehenden Links verstecken), *ExpandAllLinks* (alle ausgehenden Links anzeigen) *SearchLinks* (Versteckte Links druchsuchen und einzeln anzeigen können, die Suche soll sowohl den Titel des Ziel ***Nodes*** als auch das Label des Links verwenden) Das Suchfeld wird vom ***ikc-core*** geliefert werden.
S13	Links können separat ausgewählt und ausgeblendet werden.
S14	Die Labels der Visualisierung sollen versteckt oder angezeigt werden können.
S15	Die Visualisierung soll in den bestehenden ***ikc-core*** integriert werden.
S16	Änderungen in der Visualisierung sollen in den ***ikc-core*** als auch umgekehrt vom ***ikc-core*** in die Visualisierung übernommen werden.
S17	***Views*** sollen dauerhaft gespeichert werden. Ebenfalls sollen neue erstellt werden können.
S18	Es müssen die folgenden Dialoge bereitgestellt werden, *SearchExistingNodeToConnect*, *NewNodeToConnect* und *NewNode*.
S19	Es müssen zwei Suchfelder zur Verfügung gestellt werden: (*NodeSearch* und *LinkSearch*)

Tabelle A.2: User Stories Beschreibung

A.3 Wichtigste Protokolle

Protokoll 21.09.2016

“Kick Off”

Anwesende: Michael Kaufmann, Andreas Waldis, Patrick Siegfried

Die erste Version der Aufgabenstellung für das PAWI liegt vor. Die Grundsätze sind allen bekannt, es wird hauptsächlich auf wichtige Punkte eingegangen. Der zu erarbeitende Prototyp soll nach dem Ansatz ‘mobile-first’ entwickelt werden. Der Schwerpunkt liegt auf einer intuitiven, benutzerfreundlichen Bedienung auf Touchscreen-, wie auch auf herkömmlichen Monitoren. Die Priorität ist folgendermassen absteigend festgelegt: Mobile, Tablet, Laptop und Desktop.

Die Anforderungen werden bis zur nächsten Sitzung in der nächsten Woche gesammelt. Anschliessend werden diese besprochen und in den Anforderungskatalog aufgenommen.

Das Lösungskonzept soll das Vorgehen der Implementierung konzeptionell darlegen und so die Grundlage derer bilden. Es soll aber trotzdem agil behandelt werden, es können während der eigentlichen Entwicklung auch noch Änderungen vorkommen. Gute Inspiration für eine Visualisierung und deren Umsetzung sind der neo4j Browser aber auch mysql-Workbench. Es ist weiter noch zu ermitteln, wie die Graph-Darstellung innerhalb des react-Prototypen integriert werden soll. Um das gesamte Netzwerk übersichtlich darzustellen, ist es auch möglich nur Teile davon anzuzeigen. Und nur gegebenenfalls weitere Knoten zu laden. Das Konzept soll möglichst kurz gehalten werden.

Implementierung:
Der Programmcode (Funktionen und Parameter) ist in Prosa zu kommentieren. Das Framework wird spezifisch für den IKC Prototypen entwickelt. Die Grenzen der Schnittstellen sind noch festzulegen. Beispielsweise soll es im Prototypen möglich sein, aus der Suchkomponente gefundene Knoten direkt in die Graph-Visualisierung zu ziehen (Drag and Drop).

Projektmanagement Meilenstein: Da kein der beiden vorgeschlagenen Daten passt, hat Michael eine Mail direkt an Jörg Hofstett versandt, um das weitere Vorgehen direkt zu klären.

Wichtige Daten:
95% des Projektes in der vorletzten Woche
Rapport Abgabe: Ende Januar
Präsentation: Ende Prüfungsphase (Anfangs Februar)

Anhang: Zweite Version der Aufgabenstellung

Protokoll 21.09.2016
After “Kick Off”

Anwesende: Andreas Waldis, Patrick Siegfried

Baldmöglichst werden das Projektsetup aufgebaut und erste Anforderungen gesammelt. Für die agile Planung werden verschiedene Tools getestet, es soll eine effiziente Planung und gleichzeitig eine ausreichende Dokumentation ermöglichen. Als Datenaustausch wird Google Drive benutzt, da dort schnell und einfach an Dateien gearbeitet werden kann. Für die Dokumentation wird Latex auf der Plattform sharelatex eingesetzt.

Eine erste Version der Projektplanung wurde besprochen und in Form eines Rahmenplans gesammelt. Ergänzend dazu wurde ein Projektstrukturplan erstellt, worin die wichtigsten Etappen und Teile des Projektes zu finden sind.

Bis zu der nächsten Sitzung in der Woche vom 26.09 werden Anforderungen gesammelt.

Protokoll 26.09.2016
Anforderungen

Anwesende: Michael Kaufmann, Andreas Waldis, Patrick Siegfried

Eine erste Version der Anforderungen wurde mit dem Projektpartner besprochen. Dabei konnten diverse Punkte genauer erläutert werden. Die Integration in den bestehenden IKC-Core konnte detailliert festgelegt werden.

Die Visualisierung soll nicht automatisch geschehen, sondern wird manuell vom User vorgenommen. Es ist keine andere Darstellungsweise. Die Visualisierung bildet ein weiteres Werkzeug für den Umgang mit den Daten. Beispielsweise können die Knoten und Kanten grafisch angeordnet / gruppiert werden. Auch ist es möglich verschiedene Sichten zu denselben Daten zu erstellen. Dies bedeutet, dass eine relative Position zu den Nodes und Edges pro View mitgespeichert werden muss.

Die bestehende Node-Ansicht im ikc-core wird sich zu einer Art Detailansicht entwickeln. Dort können jeweils genauere Informationen entnommen werden, welche in der Visualisierung nicht (direkt) ersichtlich sind.

Protokoll 04.10.2016
Anforderungen

Anwesende: Michael Kaufmann, Andreas Waldis, Patrick Siegfried

Das heutige Ziel ist eine definitive Formulierung der Anforderungen im Sinne des Projektes. Dazu wird auch das Projektmanagement besprochen.

Zwischen Phasen- und Rahmenplan besteht noch eine Diskrepanz in den Begrifflichkeiten. Diverse Anpassungen der Anforderungen wurden entsprechend gemacht. Diese sind in der Projektdokumentation vorzufinden.

Es fehlt noch eine Aufstellung der Lieferobjekte und ein schlankes Risikoanalyse.
Das Projekt ist ein Forschungsprojekt + Softwareentwicklung.

Protokoll 14.10.16
Anforderungen

Anwesende: Michael Kaufmann, Andreas Waldis, Patrick Siegfried

Ziel der Sitzung: Absegnen der Ziele und Anforderungen
Alle Änderungen besprochenen Änderungen werden umgesetzt.
Es wurde beschlossen, dass mindestens bei jedem Meilenstein ein Meeting / Coaching stattfinden wird. Bei Fragen steht Michael immer zur Verfügung. Nächste Woche wird zusammen die Priorisierung der Anforderungen vorgenommen. Andreas und Patrick arbeiten am Lösungskonzept weiter.

A.4 Arbeitsjournal

Stunden Andreas Waldis

Datum	Stunden	Arbeitsschritt / Thema
21.09.2016	4	Kick Off, Sitzung, Skype Grobplanung, weiteres Vorgehen
22.09.2016	4	Setup Projekt MGMT
23.09.2016	3	Setup Projekt MGMT
26.09.2016	4	Workshop + Anpassungen Anforderungen
1.10.2016	3	Besprechung skype + Report
4.10.2016	2	Sitzung + Anpassungen
5.10.2016	1	Projektmanagement
6.10.2016	4	Projektmanagement
7.10.2016	3	Besprechugn und Anpassungen
12.10.2016	3	Konzeption
13.10.2016	3	Konzeption
14.10.2016	3	Besprechug und Konzeption
17.10.2016	3	Konzeption und Recherche
18.10.2016	5	Besprechug, Recherche und Konzeption
19.10.2016	2	Besprechung
22.10.16	1	Dokumentation
24.10.16	3	Cytoscape
26.10.16	3	Recherche + Testen
27.10.16	1	Latex-> Kriterienkatalog
04.11.16	2	Mehr Frameworks
11.11.16	4	Cytoscape in ikc-visual
16.11.16	2	Anpassungen Cyotscape
17.11.16	2	Doku
24.11.16	1	Interfaces
30.11.16	2	Besprechung, kleine Anpassungen
12.12.16	2	Besprechung
22.12.16	8	Hauptimplementierung Visualisierung
26.12.16	8	Hauptimplementierung Visualisierung
27.12.16	8	Hauptimplementierung Visualisierung
28.12.16	7	Hauptimplementierung Visualisierung + Besprechun
29.12.16	6	Hauptimplementierung Visualisierung
30.12.16	8	Hauptimplementierung Visualisierung
7.1.17	4	Hauptimplementierung Visualisierung
9.1.17	2	Hauptimplementierung Visualisierung
10.1.17	8	Hauptimplementierung Visualisierung
12.1.17	8	Integration ikc-core
13.1.17	9	Integration ikc-core
14.1.17	7	Integration ikc-core
18.1.17	3	Integration ikc-core
19.1.17	8	Integration ikc-core
20.1.17	2	Integration ikc-core / Dokumentation
23.1.17	6	Integration ikc-core / Dokumentation
24.1.17	3	Integration ikc-core / Dokumentation
25.1.17	6	Integration ikc-core / Dokumentation

Stunden Andreas Waldis

26.1.17	9	Integration ikc-core / Dokumentation
27.1.17	5	Integration ikc-core / Dokumentation
29.1.17	9	Integration ikc-core / Dokumentation
30.1.17	8	Integration ikc-core / Dokumentation

Stunden Patrick Siegfried

Datum	Stunden	Arbeitsschritt / Thema
21.09.2016	4	Kick Off, Sitzung, Skype Grobplanung, weiteres Vorgehen
22.09.2016	5	Layout Latex + Arbeitspakete
23.09.2016	3	Latex, Skype
24.09.2016	3	Anforderungen
26.09.2016	4	Skype + Sitzung + Anpassungen
27.09.2016	3	Anpassungen, Protokoll
4.10.2016	5	Projektmgmt + Sitzung + Anpassungen
5.10.2016	3	Besprechung
7.10.2016	3	Schnittstellen
10.10.2016	4	Schnittstellen
14.10.2016	3	Sitzugen + Besprechung + Anpassungen
17.10.2016	4	Journal + Anpassungen + Skype
18.10.2016	5	Latex Schnittstellen
19.10.2016	4	Sitzung + Anpassungen + Nachbesprechungen
22.10.16	2	Notizen, Latex
24.10.16	6	Cytoscape
25.10.16	4	Recherche
27.10.16	3	Latex-> Kriterienkatalog
04.11.16	4	Mehr Frameworks
11.11.16	4	Cytoscape in ikc-visual
16.11.16	5	Anpassungen Cyotscape
17.11.16	2	Doku
24.11.16	5	Interfaces
30.11.16	2	Besprechung, kleine Anpassungen
12.12.16	4	Interfaces
28.12.16	5	Besprechung vor Ort
02.01.16	2	UI/UX
03.01.16	8	UI/UX
04.01.16	5	Integration
07.1.16	8	expand-collapse
8.1.16	8	expand-collapse + git
9.1.16	4	Integration
10.1.16	3	Integration
11.1.16	5	Integration
13.01.16	4	ikc-visual
14.01.16	3	ikc-visual
15.01.16	3	ikc-visual
18.1.16	4	Besprechung + Anpassungen
20.01.16	3	Details UI
21.01.16	4	ikc-visual
22.1.16	4	Doku
23.1.16	5	Besprechung + Anpassungen UI

Stunden Patrick Siegfried

24.1.16	3	Integration
25.1.16	2	Besprechung
27.1.16	5	Integration
28.1.16	4	Doku
29.1.16	8	Doku
30.1.16	8	Doku + Abschluss

A.5 Aufgabenstellung

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Rotkreuz, 19. September 2016
Seite 1/5

Abteilung Informatik

Informatikprojekt

Aufgabe für
- Patrick Siegfried
- Andreas Waldis

1. Arbeitstitel (kurz und aussagekräftig)

Prototyping einer webbasierten Visualisierung zur intuitiven Interaktion mit einem Wissensnetzwerk

2. Fachliche Schwerpunkte (max. 5 Stichworte)

Wissensnetzwerk

User Interface Design

Visuelle Interaktion

Drag'n'Drop

Web/Javascript

3. Einleitung (Hintergrund, Ausgangslage, Hinführung zur Problemstellung/Aufgabenstellung)

Im Hasler-Projekt Intuitive Knowledge Connectivity (IKC) wird ein Prototyp für ein plattformübergreifendes Wissensnetzwerk erstellt. Die grundlegende Datenbank basiert auf einem Netzwerk (einem gerichteten beschrifteten Property-Graph), hat aber bisher nur eine technische Konsole. Eine ideale grafische Benutzerschnittstelle stellt das Netzwerk oder Teilausschnitte daraus visuell und zweidimensional dar, und stellt beschriftete Knoten und Pfeile grafisch dar. Dies ermöglicht eine einfache und übersichtliche Repräsentation und eine intuitive Interaktion mit den gegebenen Kanten und Knoten. Der bestehende Prototyp implementiert die grundlegenden Datenbankoperationen (CRUD) und die Verknüpfung von Knoten mit Drag and Drop. Auf dieser Kern-Software soll aufgebaut werden, um diese hinsichtlich intuitiver Benutzung zu erweitern, damit mit mehreren Knoten im Netzwerk gleichzeitig und visuell gearbeitet werden kann. Dies soll in erster Linie auf dem Touchscreen (mobil / Tablet) und in zweiter Linie

Hochschule Luzern
Technik & Architektur

responsive mit dem gleichen Code auch mit Bildschirm, Tastatur und Maus / Touchpad bedienbar sein.

4. Aufgabenstellung (konkrete Aufgaben/Zielsetzungen [inhaltlich], allenfalls Hinweise zur Vorgehensweise)

Für eine optimale Visualisierung eines Graphen existieren diverse bestehende Bibliotheken und Grundstrukturen. Auch gibt es verschiedene Ansätze, dieses Vorhaben komplett eigenhändig zu implementieren. Bis zum jetzigen Zeitpunkt ist es nicht geklärt, welche Variante den Anforderungen, insbesondere responsive & mobile friendly, am besten gerecht wird. Darum soll dies in einem ersten Teil kurz geklärt werden. Dazu müssen bestehende Varianten für APIs / Libraries gesammelt und anschliessend, falls nötig, kurz mit einem minimalen Demonstrator evaluiert werden. Als zweiter, wesentlich grösserer Teil soll aus den Resultaten eine voll funktionsfähige responsive und mobile-fähige Benutzerschnittstelle umgesetzt werden. Diese ermöglicht dem Benutzer die Anwendung der grundlegenden Datenbankoperationen (CRUD) und zudem visuell ein Drag & Drop fähiges, zweidimensionales Netzwerk mit visueller Interaktion. Als Vision kann das Interface der Neo4J Datenbank herangezogen werden, welche um die Möglichkeit für CRUD direkt im visuellen Graph angereichert wird.

Die zu entwickelnde Benutzerschnittstelle wird aufbauend auf und integriert mit dem bestehenden Prototypen IKC entwickelt. Die neue Schnittstelle kann in einer eigenen Javascript-Library gekapselt werden. Ziel ist ein live-Demo einer visuellen Interaktion mit technischem Durchstich für CRUD bis zum IKC-Core.

Lieferobjekte:

- Anforderungskatalog
- Lösungskonzept inkl. Schnittstellendefinition
- Funktionale Software mit folgenden Eigenschaften:
  - Integriert bestehenden Prototyp auf Basis React mit allen CRUD Funktionen & Schnittstellen zu Dropbox und Evernote
  - Gibt die Möglichkeit zur visuellen Interaktion mit einem (Teil-)Netzwerk: Knoten können per Drag & Drop verknüpft werden
  - Die Applikation läuft auf dem Mobile, Tablet, Laptop und Desktop (in Priorität der Reihenfolge) => responsive CSS Template verwenden
- Dokumentierter Source Code (für Methoden und Parameter)
- Benutzerhandbuch
- PAWI-Bericht

Die Projektarbeit PAWI umfasst folgende Teilaufgaben:

1.) Die Projektplanung
2.) Erstellung einer Liste mit Anforderungen

Hochschule Luzern
Technik & Architektur

3.) Gestaltung eines Lösungskonzepts
4.) Implementierung eines Prototyps / Demonstrators
5.) Schreiben des Abschlussberichts inkl. Benutzerhandbuch

5. Durchführung der Arbeit (z.B. Hinweise betr. Projektplan, Terminen, Zwischenberichten, Organisation)

Während der Projektarbeit ist ein persönliches Arbeitsjournal zu führen und dem/der Dozenten/in regelmässig zukommen zu lassen. Ein Arbeitsjournaleintrag beinhaltet zumindest Datum, Anzahl Stunden, Arbeitsschritt/Thema und allenfalls Bemerkungen (Probleme, Fragestellungen, Zwischenergebnisse…). Das Arbeitsjournal ersetzt nicht die Projektplanung.

Projekt-Vorgehen:

- Planung, Organisation, Risikomanagement sind Teil der Aufgabe und werden von den Studierenden wahrgenommen.
- Bis spätestens Semesterwoche 3 müssen Sie entscheiden können, zu welcher Projektart ihre PAWI-Arbeit gehört (siehe Ilias Ordner Organisation, PAWI-BDA-Vorgehen), eine passende Vorgehensart wählen und einen entsprechenden Rahmenplan/Grobplan (Vorgehen und Planung) erarbeitet haben.
  Am besten skizzieren Sie dazu initiale Teile des Grobkonzept (vgl. Modul IM) soweit, dass Ihre Überlegungen nachvollziehbar sind und am Coaching diskutiert werden können.
- Die Studierenden nehmen an einem der obligatorischen Projekt-Coachings ca. in Semesterwoche 3 (Termine werden kommuniziert) teil und bringen den Entwurf ihres Grobkonzepts elektronisch mit, damit er im Plenum besprochen und diskutiert werden kann (Beamer).

→ Beginnen Sie zu Beginn mit der Erstellung eines individuellen *Projektplans*.
Am Anfang der Arbeit wird das Gesamtprojekt unterteilt in Arbeitspakete (bzw. Epics), Meilensteine (bzw. Sprints) und Lieferobjekte (bzw. User Stories).

- *Arbeitspakete* untereilten die Arbeit in sinnvolle Einheiten.
- *Lieferobjekte* spezifizieren konkrete Ergebnisse, also entweder Dokumente oder Software
- *Meilensteine* definieren, zu welchem Zeitpunkt ein wichtiger Arbeitsschritt erreicht ist ist, also beispielsweise die Fertigstellung eines Arbeitspakets, die Abnahme eines Lieferobjekts, oder die Durchführung eines wichtigen Meetings.

Dieser Projektplan wird vom Studierenden zu Semesterbeginn selbstständig erstellt. Der Projektplan strukturiert anschliessend die Gesamte Durchführung des Projekts:

- Für jeden Meilenstein gibt es ein *Meilensteinmeeting* mit Auftraggeber und Dozierendem, welches vom Studierenden zu organisieren ist.
- Bis zu diesem Meeting werden die entsprechend definierten Arbeiten durchgeführt und Lieferobjekte erstellt.
- In diesem Meeting werden die entsprechenden Arbeitspakete und Lieferobjekte geprüft, korrigiert oder für gut befunden, so dass das Projekt weiter gehen kann.

Vorgabe: Drei Meilensteine sind fix:

- Das Kick-off zu Beginn des Semesters.
- Das Projekt-Coaching in der Semesterwoche 3 (Lieferobjekte: Projektplanung, erster Entwurf des Lösungskonzepts)
- Das Abschlussmeeting in der vorletzten Woche. Bis zu diesem Datum sollten 95% der Arbeiten erledigt sein, damit in der letzten Woche nur noch wenige Änderungen nötig sind

Hochschule Luzern
Technik & Architektur

Der Dozent steht für Coaching-Termine wöchentlich zur Verfügung. Diese sind freiwillig. Bei Bedarf ist es an den Studierenden, diese zu planen und zu organisieren.

6. Dokumentation

Der Schlussbericht enthält zwingend:

- die folgende Selbstständigkeitserklärung auf der Rückseite des Titelblattes:
  „Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.
  Rotkreuz, Datum, eigenhändige Unterschrift"

Zusätzlich muss dem betreuenden Dozenten / der betreuenden Dozentin ein Download-Link auf ILIAS für ein Zip-Archiv (Keine CDs / Sticks / physische Datenträger!) mit dem Bericht (inkl. Anhänge), mit den Präsentationen, Messdaten, Programmen, Auswertungen, usw. abgeben werden.

→ Erarbeiten Sie für den Schlussbericht eine einheitliche Struktur in einem einzigen Dokument. Verzichten Sie so weit wie möglich auf Anhänge. Hier ist ein Vorschlag für die Inhaltsstruktur mit acht Kapiteln:

1. *Einleitung*: Problemstellung und Ziel der Arbeit;
2. *Lösungsdesign*: Projektplanung; Anforderungsliste; Modelle (Klassen, Deployment, Komponenten usw.). Lösungskonzepte
3. *Implementation*: Beschreibung des Prototyps; Benutzerhandbuch
4. *Schlussfolgerungen*: Zusammenfassung; Erkenntnisse; Lessons Learned; Ausblick
5. *Literatur*: Quellen und verwendete Arbeiten, Texte, Software, Links, …
6. *Anhang*: Z.B. Sitzungsprotokolle, …

7. Fachliteratur/Web-Links/Hilfsmittel (optional)

Kaufmann, M., Waldis, A, Siegfried, P., Wilke, G., Portmann, E., Hemmje, M. (2016). Intuitive Knowledge Connectivity: Design and Prototyping of Cross-Platform Knowledge Graphs. In Proceedings of Knowledge Science, Engineering and Management KSEM 2016, Passau, Germany. Springer LNCS (to appear)

https://www.dropbox.com/s/irgyjtoesybg925/Preprint_IKC_KSEM16.pdf?dl=0

8. Zusätzliche Bemerkungen (optional; spezielle Vereinbarungen, Geheimhaltungserklärung, Intellectual Property, usw.)

Die HSLU erhält ein einfaches unbeschränktes Nutzungsrecht für die Ergebnisse.

Die Arbeit wird im virtuellen Team entstehen, ein Teammitglied ist in den USA im Auslandsemester.

Es ist an den Studierenden die entsprechenden Mittel/Methoden für ein solches Vorgehen einzusetzen. Der Umgang mit dieser verteilten Entwicklung wird auch Teil der Bewertung sein.

Verschiebung der Abgabe: Es ist ok, wenn die Arbeit Ende Januar abgegeben wird.

Hochschule Luzern
Technik & Architektur

9. Industrie-/Wirtschaftspartner (Kontaktpersonen)

Dr. Michael Kaufmann
Projektleiter IKC
Hochschule Luzern – Informatik
Suurstoffi 41
6343 Rotkreuz
m.kaufmann@hslu.ch

10. Verantwortlicher Dozent / verantwortliche Dozentin, Betreuungsteam

Michael Kaufmann (verantwortlicher Dozent / verantwortliche Dozentin)

Ort, Datum, Unterschrift