Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Katharina Ellermann

Eine Anwendung zur Erhebung der Datenherkunft von Kennzahlen im Rahmen des Sanierungsmanagements in Unternehmen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Katharina Ellermann

Eine Anwendung zur Erhebung der Datenherkunft von Kennzahlen im Rahmen des Sanierungsmanagements in Unternehmen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuende Prüferin: Prof. Dr. Ulrike Steffens
Zweitgutachter: Dr. Jonas Steeger

Eingereicht am: 18. Januar 2018

Katharina Ellermann

Thema der Arbeit
Eine Anwendung zur Erhebung der Datenherkunft von Kennzahlen im Rahmen des Sanierungsmanagements in Unternehmen

Stichworte
Sanierungsmanagement, Anwendungsentwicklung, Enterprise Architecture Management, Enterprise Architecture Management Pattern Catalog, Angular, REST

Kurzzusammenfassung
Im Sanierungsmanagement werden Kennzahlen für die Messung von Maßnahmeneffekten benötigt. Diese Kennzahlen verteilen sich jedoch über viele Datenquellen im Unternehmen. Die vorliegende Arbeit dokumentiert die Konzeption und Implementierung einer voll funktionsfähigen Anwendung zur Erfassung und Veranschaulichung der Datenherkunft von Kennzahlen. Hierzu wurde ein Pattern-Konstrukt in Anlehnung an den Enterprise Architecture Management Pattern Catalog entwickelt.

Katharina Ellermann

Title of the paper
An Application to Collect the Source of Data in the Context of Turnaround Management in Companies

Keywords
Turnaround Management, Application Development, Enterprise Architecture Management, Enterprise Architecture Management Pattern Catalog, Angular, REST

Abstract
In the context of Turnaround Management key performance indicators (KPIs) are needed for the illustration of measure effects. These KPIs are spreaded throughout data sources in the company. This paper documents the conception and development of a fully operating application to collect and visualize the souce of data of KPIs. A pattern consctruct based on the Enterprise Architecture Management Pattern Catalog was designed for this purpose.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Die vorliegende Arbeit dokumentiert die Konzeption und Implementierung einer Anwendung zur Erfassung und Veranschaulichung der Datenherkunft von Kennzahlen. Seinen Einsatz soll das Tool im Sanierungsmanagement haben.

> *„Kennzahlen haben die Aufgabe, aus der Flut der betrieblichen Informationen das Wesentliche herauszufiltern. Das Management oder Investoren benötigen für Entscheidungen ein Instrumentarium, das übersichtlich und in konzentrierter Form entscheidungsrelevante Informationen über die wichtigsten betrieblichen Sachverhalte liefert. Neben der Entscheidungsunterstützung helfen Kennzahlen auch bei anderen betriebswirtschaftlichen Aufgaben wie der Planung, Kontrolle, Koordination oder Motivation und Verhaltenssteuerung."*[1]

Auch im Sanierungsmanagement, also der Restrukturierung von Unternehmen in der Krise, spielen Kennzahlen eine große Rolle. Sie werden benötigt, um Maßnahmeneffekte messen zu können. Als Maßnahme wird eine konkrete Tätigkeit bezeichnet, die notwendig ist, um eine Strategie umzusetzen und Unternehmensziele zu erreichen.[2] Ein Maßnahmeneffekt ist der Effekt, den eine geplante Restrukturierungsmaßnahme auf die künftige Entwicklung von Vermögens-, Finanz- oder Ertragslage des Unternehmens hat.[3]

Allerdings sind die Kennzahlen, die für die Messung der Maßnahmeneffekte erforderlich sind, oftmals über verschiedenste Informationssysteme verstreut oder können

[1] Vgl. Losbichler, Eisl und Engelbrechtsmüller, *Handbuch der betriebswirtschaftlichen Kennzahlen: Key Performance Indicators für die erfolgreiche Steuerung von Unternehmen*, S. v.

[2] Vgl. Doppler und Lauterburg, *Change Management – Den Unternehmenswandel gestalten*, S. 209.

[3] Vgl. Institut der Wirtschaftsprüfer e.V., *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*, Rz. 135f.

aus Excel-Tabellen stammen. Kaum jemand hat eine Übersicht, wie aktuell welche Kennzahl ist oder in welchem Abstand sie aktualisiert wird. Unklar ist oft auch, ob eine Schnittstelle zu dem Informationssystem, in dem die Kennzahl liegt, vorhanden ist und um was für eine Schnittstelle es sich handelt.

Beratungsunternehmen, die sich kriselnder Unternehmen annehmen, stehen immer wieder vor der zeit- und arbeitsintensiven Aufgabe, sich die benötigten Kennzahlen mühsam zusammensuchen zu müssen. Fehlende Dokumentation und Unwissen innerhalb der Unternehmen verschärfen diese Situation.

Vor diesem Hintergrund hat diese Arbeit das Ziel, ein Instrument zu schaffen, mit dessen Hilfe Unternehmen sich einen Überblick über die Datenherkunft ihrer Kennzahlen verschaffen können. Die Anwendung soll zum einen die Möglichkeit bereithalten, Datenquellen, Kennzahlen, Maßnahmen und Effekte zu erfassen und zum anderen eine visuelle Darstellung der Zusammenhänge zwischen diesen Entitäten bieten.

1.1. Kooperationspartner

Die vorliegende Arbeit wurde in Zusammenarbeit mit der Nordantech GmbH & Co. KG erstellt. Die Nordantech GmbH & Co. KG ist ein im Mai 2016 gegründetes Startup, das die Software Falcon entwickelt und vertreibt. Bei Falcon handelt es sich um ein webbasiertes multi-user Projektmanagementtool für Change Projekte im Business Transformation Bereich. Dazu gehören Restrukturierungen, Turnarounds und Unternehmenskäufe. Falcon bietet Unterstützung dabei, diese Projekte zu strukturieren, zu verfolgen und voranzutreiben.

1.2. Aufbau der Arbeit

Kapitel 2 befasst sich mit den theoretischen Grundlagen: Zunächst wird die fachliche Domäne, das Sanierungsmanagement, der Anwendung beschrieben. Ein grundlegendes Verständnis des Sanierungsmanagements und somit der Einsatzumgebung der Anwendung ist die Voraussetzung, um eine sorgfältige Anforderungsanalyse durchzuführen.

Weiterhin wird der Enterprise Architecture Management Pattern Catalog[4] vorgestellt, auf dessen Basis im späteren Verlauf der Arbeit ein eigenes Pattern-Konstrukt entwickelt wird, um eine passende Visualisierung bieten zu können.

In Kapitel 3 wird eine Anforderungsanalyse durchgeführt und der Funktionsumfang der Anwendung beschrieben. Hierzu werden Stakeholder identifiziert, Use Cases entwickelt und sowohl funktionale als auch nichtfunktionale Anforderungen festgehalten.

Kapitel 4 befasst sich mit der Konzeption der Patterns und der Software. Der Softwareentwurf beruht auf den in Kapitel 3 identifizierten Anforderungen.

Kapitel 5 beschreibt die Implementierung der Anwendung auf Basis des Softwareentwurfs.

In Kapitel 6 werden die Ergebnisse der Arbeit zusammengefasst und weiterhin ein Ausblick gegeben, wie diese Anwendung in der Praxis genutzt und um zusätzliche Funktionen erweitert werden kann.

[4] Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*.

2. Theoretische Grundlagen

In diesem Abschnitt werden die theoretischen Grundlagen für die folgenden Kapitel erklärt. Hierzu wird zunächst das Sanierungsmanagement vorgestellt und auf die Unternehmenskrise als Anwendungsfeld des Sanierungsmanagements und die verschiedenen Krisenstadien eingegangen. Außerdem werden das Sanierungskonzept nach dem „IDW[1]-Standard: Anforderungen an die Erstellung von Sanierungskonzepten“ (IDW S 6) und Sanierungsmaßnahmen erläutert. Auf diese Weise wird die zu entwickelnde Anwendung in ihre fachliche Domäne eingebettet und das Konzept von Maßnahmen und Maßnahmeneffekten veranschaulicht.

Danach wird das Enterprise Architecture Management und der Enterprise Architecture Pattern Catalog (EAMPC) näher betrachtet. Zunächst werden die verschiedenen Elemente des Pattern Catalogs kurz vorgestellt und anschließend anhand eines Beispiels aus dem Katalog näher erläutert. Die Patterns aus dem EAMPC sollen im weiteren Verlauf der Arbeit als Grundlage für die Entwicklung eines eigenen Pattern-Konstruktes dienen.

2.1. Sanierungsmanagement

2.1.1. Unternehmenskrise

Als Unternehmenskrise wird eine Situation bezeichnet, in der die Existenz eines Unternehmens nicht mehr gesichert ist.[2] Es handelt sich hierbei in den meisten Fällen um einen langsamen Prozess, dessen Ursachen über lange Zeit unentdeckt bleiben, bis

[1] Institut der Wirtschaftsprüfer in Deutschland e.V.

[2] Vgl. Institut der Wirtschaftsprüfer e.V., *WP Handbuch 2008: Wirtschaftsprüfung, Rechnungslegung, Beratung, Band II*, F Rz. 21.

die Symptome schließlich sichtbar werden und die Gefahr einer Insolvenz aufkommt.[3] Je früher das Stadium der Unternehmenskrise, in der diese entdeckt wird, desto größer sind die Chancen einer erfolgreichen Bewältigung dieser Krise. Sie kann sogar genutzt werden, um historisch gewachsene, ungesunde Strukturen aufzubrechen und Veränderungen anzustoßen.[4]

Der IDW S 6[5] unterscheidet sechs unterschiedliche Krisenstadien, die nicht zwingend voneinander abgegrenzt nacheinander ablaufen müssen, sondern auch in anderer Reihenfolge, gleichzeitig oder sich überschneidend, verlaufen können:

- Stakeholderkrise
- Strategiekrise
- Produkt- und Absatzkrise
- Erfolgskrise
- Liquiditätskrise
- Insolvenzreife

In Abbildung 2.1 ist ein typischer Krisenverlauf mit Entwicklung des Ertrags und der Liquidität des Unternehmens dargestellt.

[3] Vgl. Hohberger und Damlachi, *Praxishandbuch Sanierung im Mittelstand*, S. 1.
[4] Vgl. Institut der Wirtschaftsprüfer e.V., *WP Handbuch 2008: Wirtschaftsprüfung, Rechnungslegung, Beratung, Band II*, F Rz. 32.
[5] Vgl. Institut der Wirtschaftsprüfer e.V., *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*, Rz. 62.

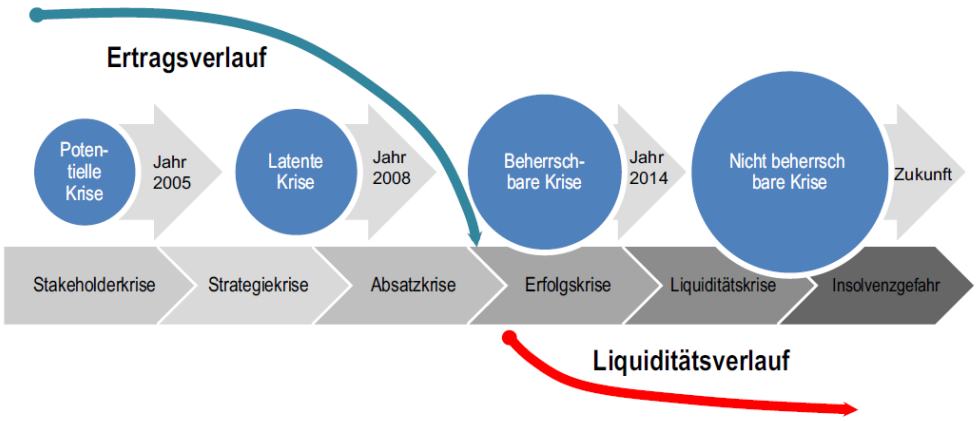

Abbildung 2.1.: Typischer Krisenverlauf (Quelle: Hohberger und Damlachi, *Praxishandbuch Sanierung im Mittelstand*, 2014)

Nachfolgend werden die einzelnen Krisenstadien näher erläutert.

Stakeholderkrise

Mit einer Stakeholderkrise beginnt eine Unternehmenskrise typischerweise. Als Stakeholder werden alle Personen oder Personengruppen bezeichnet, die irgendein Interesse an dem Verlauf oder dem Ergebnis eines Prozesses haben.[6] Im Unternehmenskontext sind dies beispielsweise Gesellschafter, das Management, die Arbeitnehmer und ihre Interessenvertreter, Kunden und Kreditgeber.[7]

Die Stakeholderkrise zeichnet sich durch ein nachlässiges Führungsverhalten aus, das in Meinungsverschiedenheiten zwischen oder innerhalb der verschiedenen Stakeholdergruppen begründet ist. Effizienzverluste und die Verzögerung wichtiger Entscheidungen sind die Folge. Der Belegschaft fehlt ein klares Unternehmensleitbild, was zur Abnahme von Motivation und Leistungsbereitschaft führt.[8]

Strategiekrise

Die in der Stakeholderkrise beginnende Nachlässigkeit im Führungsstil resultiert in einer unklaren strategischen Ausrichtung. Vorhandene Erfolgspotenziale werden

[6] Vgl. Eilmann et al., „Interessengruppen/Interessierte Parteien“, S. 71.
[7] Vgl. Crone, „Die Unternehmenskrise“, S. 5.
[8] Vgl. Groß, „Erkennen und Bewältigen von Unternehmensschieflagen“, S. 129.

nicht ausgeschöpft oder sind verbraucht und es werden keine neuen Erfolgspotenziale geschaffen.[9]

Produkt- und Absatzkrise

Eine Produkt- und Absatzkrise zeichnet sich durch eine sinkende Nachfrage nach den Gütern bzw. Dienstleistungen eines Unternehmens aus. Das Unternehmen zeigt Schwächen in Qualität, Marketing und Vertrieb, zudem konzentriert es sich nicht ausreichend auf diejenigen Kunden und Produkte, die positive Deckungsbeiträge bringen. Auf die Produkt- und Absatzkrise folgt unvermeidlich eine Erfolgskrise, wenn nicht schon zu diesem Zeitpunkt mit entsprechenden Maßnahmen gegengesteuert wird.[10]

Erfolgskrise

Während einer Erfolgskrise leidet ein Unternehmen stark unter Verlusten oder sinkenden Gewinnen, die das Eigenkapital stetig minimieren. Eine Gefahr der Insolvenz muss noch nicht vorliegen, allerdings ist das Unternehmen bereits nicht mehr in der Lage, die Mittel für eine Sanierung selbst zu tragen. Dieses Kapital müsste durch Dritte aufgebracht werden.[11]

Liquiditätskrise

Die Liquiditätskrise ist gekennzeichnet durch „die konkrete und akute Gefahr der Zahlungsunfähigkeit“.[12] Die Schlüsselfaktoren, die in der Vergangenheit zum Erfolg des Unternehmens führten, sind nicht mehr im benötigten Umfang vorhanden und eine ungünstige Finanzstruktur liegt vor.[13]

[9] Vgl. Hohberger und Damlachi, *Praxishandbuch Sanierung im Mittelstand*, S. 13.
[10] Vgl. Institut der Wirtschaftsprüfer e.V., *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*, Rz. 73.
[11] Vgl. Institut der Wirtschaftsprüfer e.V., *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*, Rz. 74f.
[12] Crone, „Die Unternehmenskrise“, S. 7.
[13] Vgl. Institut der Wirtschaftsprüfer e.V., *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*, Rz. 78.

Insolvenzreife

Wurden in den zuvor beschriebenen Stadien der Unternehmenskrise keine Maßnahmen getroffen, die dem entgegenwirken, droht die Insolvenzreife. Eine betriebswirtschaftliche Krise geht in eine insolvenzrechtlich relevante Krise über, wenn einer der folgenden Insolvenzgründe vorliegt:

- (eingetretene) Zahlungsunfähigkeit
- drohende Zahlungsunfähigkeit
- Überschuldung

Der Zustand der Zahlungsunfähigkeit tritt ein, wenn ein Schuldner nicht in der Lage ist, seine fälligen Zahlungspflichten zu erfüllen.[14] Selbst ein Zahlungsverzug von 14 Tagen, der in dem Fehlen von liquiden Mitteln begründet liegt, kann den Zustand einer Zahlungsunfähigkeit herstellen.

Eine drohende Zahlungsunfähigkeit gemäß §18 Insolvenzordnung (InsO) hingegen tritt bereits dann ein, wenn ein Unternehmen „voraussichtlich nicht in der Lage sein wird, die bestehenden Zahlungspflichten im Zeitpunkt der Fälligkeit zu erfüllen."

Gemäß §19 InsO liegt eine Überschuldung vor, „wenn das Vermögen des Schuldners die bestehenden Verbindlichkeiten nicht mehr deckt, es sei denn, die Fortführung des Unternehmens ist nach den Umständen überwiegend wahrscheinlich".

2.1.2. Sanierungskonzept nach IDW S 6

Befindet sich ein Unternehmen in einer Krisensituation, muss die Geschäftsführung handeln, um den bestehenden Risiken entgegenzuwirken. Hierzu wird ein Sanierungskonzept erstellt, das als Entscheidungsgrundlage für alle Beteiligten dient und die Sanierungsfähigkeit des Unternehmens beurteilt. Sanierungsfähig ist ein Unternehmen, „wenn nicht nur die Fortführungsfähigkeit i.S.d. §252 Abs. 1 Nr. 2 Handelsgesetzbuch (HGB) bejaht, sondern wenn darüber hinaus nachhaltig die Wettbewerbsfähigkeit und Renditefähigkeit [...] wieder erlangt werden kann".[15]

[14] Vgl. Crone und Werner, „Rechtliche Rahmenbedingungen und Prüfung der Insolvenztatbestände", S. 17ff.

[15] Crone, „Erstellung von Sanierungskonzepten nach IDW S 6", S. 62.

Inhaltlich umfasst ein Sanierungskonzept wichtige Unternehmensdaten, rechtliche und ökonomische Rahmenbedingungen, die Maßnahmen, die zur Bewältigung der Krise umzusetzen sind, sowie die Quantifizierung der Maßnahmeneffekte.[16]

Das Konzept wird entweder durch die Geschäftsleitung selbst, mit Beteiligung externer Berater oder gänzlich durch externe Berater erstellt. Je weiter vorangeschritten die Krise ist, desto stärker entscheidet dies die finanzierende Institution.

2.1.3. Sanierungsmaßnahmen

Sanierungsmaßnahmen werden unter sogenannten Stoßrichtungen zusammengefasst, die als Haupttreiber der Sanierung gelten. Die Stoßrichtungen bauen auf Wettbewerbsvorteilen im Markt auf, also auf dem Verfügen über bestimmte Differenzierungsmerkmale (z.B. Kostenführerschaft) durch die strategische Positionierung des Unternehmens Ein Beispiel dafür ist der Ausbau eines profitablen Segments oder die Bereinigung des Produktportfolios durch den Verkauf bestimmter Aktivitäten.

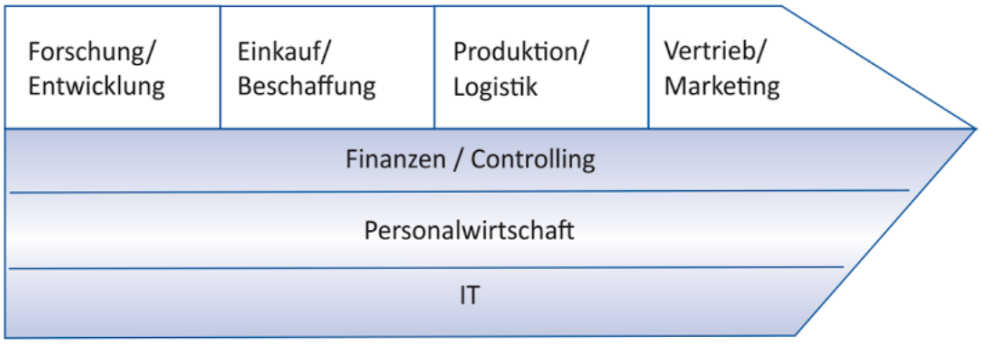

Abbildung 2.2.: Wertschöpfungskette (Quelle: Werner, „Leistungswirtschaftliche Sanierungsmaßnahmen“, 2014)

Innerhalb der Stoßrichtungen werden die Maßnahmen entlang der Wertschöpfungskette organisiert und als Maßnahmenpakete gebündelt. In Abbildung 2.2 ist die Wertschöpfungskette mit den Primäraktivitäten, die einen direkten wertschöpfenden Beitrag liefern, und den Sekundäraktivitäten, die unterstützend wirken, dargestellt.

[16] Vgl. Institut der Wirtschaftsprüfer e.V., *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*, Rz. 2.

Ein Beispiel für diese Zusammenhänge ist Folgendes:

- **Stoßrichtung:** Cash Management (Liquiditätserhöhung)
  - **Maßnahmenpaket:** Working Capital Management[17]
    - **Maßnahme 1:** Lagerbestandsreduktion
    - **Maßnahme 2:** Zahlungszielmanagement

Sanierungsmaßnahmen werden weiterhin in leistungswirtschaftliche und finanzwirtschaftliche Sanierungsmaßnahmen unterteilt.

Leistungswirtschaftliche Sanierungsmaßnahmen

Leistungswirtschaftliche Maßnahmen dienen der Verbesserung der Struktur des Unternehmens und sollen die Ursachen für die Krise beheben.

Im Folgenden wird der Bereich IT beispielhaft mit ausgewählten leistungswirtschaftlichen Sanierungsmaßnahmen vorgestellt[18]:

- **Prüfung und Verbesserung der Stammdatenqualität**
  Um Berechnungen im Rahmen der Sanierungsplanung durchführen zu können, muss eine ausreichende Qualität der Stammdaten gesichert sein.
- **Analyse der IT-Leistungsfähigkeit und -Kosten**
  Eine heterogene Geräte- und Softwareausstattung kann hohe Kosten verursachen. Eventuell können durch eine Umstellung Kosten eingespart und eine bessere Unterstützung der Geschäftsprozesse realisiert werden.
- **Analyse der vorhandenen Systeme und Projekte**
  Bestehende Systeme und Projekte sollten auf Betriebsnotwendigkeit geprüft werden und ggf. abgeschaltet bzw. nicht weiter verfolgt werden, um Kosten einzusparen.

[17] Working Capital = Umlaufvermögen – kurzfristige Verbindlichkeiten
[18] Vgl. Werner, „Leistungswirtschaftliche Sanierungsmaßnahmen“, S. 128.

Finanzwirtschaftliche Sanierungsmaßnahmen

Die Unternehmensfinanzierung ist in der Sanierung ein äußerst wichtiger Baustein. Mit Voranschreiten der Krise wächst auch die Bedeutung der Liquiditätsbeschaffung. In der Strategiekrise beispielsweise verfügt ein Unternehmen oftmals noch über ausreichende Liquidität, während in der Erfolgskrise bereits Verluste ausgeglichen werden müssen.

Kurzfristiges Ziel der Unternehmensfinanzierung ist die Sicherstellung ausreichender Liquidität zur Geschäftsfortführung. Doch auch die Umsetzung des Sanierungsplans muss finanziert werden, um die langfristige Sanierung des Unternehmens zu ermöglichen.

Bei der Unternehmensfinanzierung wird in Innen- und Außenfinanzierung unterschieden. Die Innenfinanzierung betrifft alle Finanzierungsmöglichkeiten, die ein Unternehmen aus eigener Kraft bereitstellen kann. Eine Möglichkeit hierfür wäre die Freisetzung bestehender Liquiditätsreserven oder der Verkauf von nicht betriebsnotwendigem Vermögen. Die Außenfinanzierung besteht in der Erhöhung des Eigen- oder Fremdkapitals, beispielsweise durch externe Investoren oder ein Bankdarlehen.[19]

[19] Vgl. Hettich, Kreide und Crone, „Finanzwirtschaftliche Sanierungsmaßnahmen“, S. 129-149.

2.2. Enterprise Architecture Pattern Catalog

Aufgrund sich schnell ändernder Anforderungen und Marktanpassungen stehen Unternehmen vor der Herausforderung einer stetig wachsenden Komplexität ihrer IT-Systemlandschaft. Gleichzeitig verändert sich die Rolle der IT im Unternehmen vom einfachen Dienstleister hin zu einem wichtigen Treiber für neue Geschäftsmodelle. Enterprise Architecture Management (EAM), also das Management der Unternehmensarchitektur, versucht, die IT und das Business zu harmonisieren und so Kosteneinsparungspotenziale zu nutzen und gleichzeitig die Verfügbarkeit und die Fehlertoleranz von IT-Systemen zu verbessern.[20] Weiterhin kann die IT mithilfe von EAM die Entwicklung neuer Geschäftsmodelle ermöglichen und aktiv mitgestalten.[21]

Um den wachsenden Anforderungen an das EAM in der Praxis gerecht zu werden, hat die Forschungsstelle sebis[22] der Technischen Universität München ihre Forschungsergebnisse und Erfahrungen aus Projekten in einem EAM Pattern Catalog veröffentlicht. Dieser enthält Best Practises und Patterns für unterschiedlichste Anwendungsgebiete des EAM.[23]

[20] Vgl. Roth et al., „Enterprise Architecture Documentation: Current Practices and Future Directions“, S. 1f.
[21] Vgl. Hanschke, *Enterprise Architecture Management – einfach und effektiv*, S. 38.
[22] Software Engineering for Business Information Systems
[23] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 1.

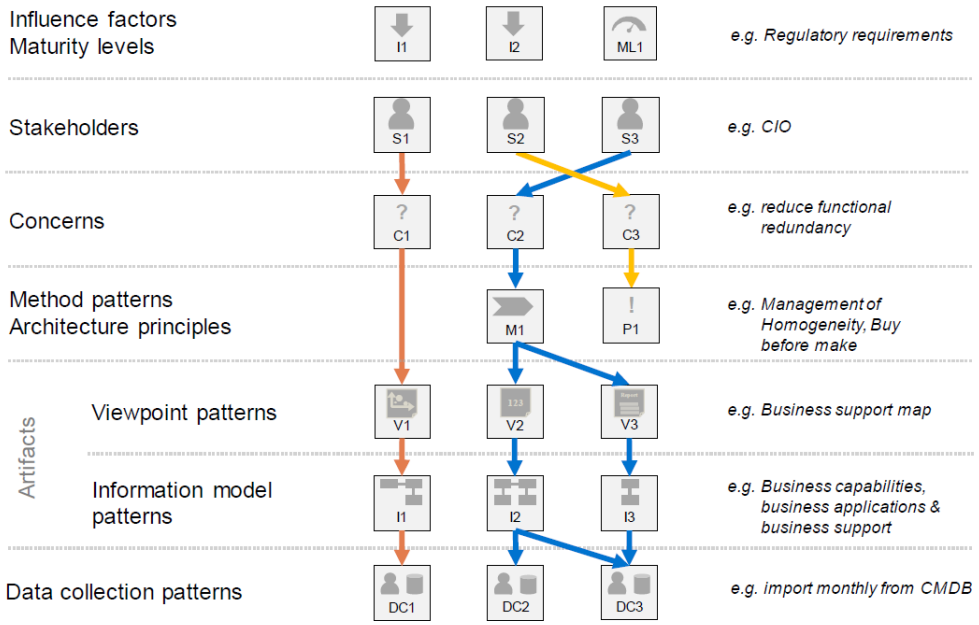

Abbildung 2.3.: Erweiterte Sprache des EAMPC mit Beispielen (Quelle: Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, 2015)

In Abbildung 2.3 sind alle Elemente der EAMPC-Sprache dargestellt und werden im Folgenden kurz erläutert[24]:

- **Einflussfaktoren:**
  Einflussfaktoren beschreiben den Unternehmenskontext, der den EAM-Prozess beeinflussen kann (z.B. regulatorische Anforderungen). Je nach Einflussfaktor kann ein EAM-Pattern mehr oder weniger bedeutend sein.

- **Stakeholder:**
  Das Management einer Unternehmensarchitektur kann nur durch Zusammenarbeit bewältigt werden. Dafür ist es wichtig, die wesentlichen Stakeholder an der Unternehmensarchitektur zu kennen.

- **Anliegen (Concern):**
  Die Concerns beschreiben die Interessen von Stakeholdern, die eigene Ziele für die Unternehmensarchitektur haben.

[24] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 5f.

- **Method Patterns:**
  Method Patterns beschreiben die Schritte, die notwendig sind, um die Anliegen der Stakeholder umzusetzen.

- **Viewpoint Pattern:**
  Viewpoint Patterns veranschaulichen wesentliche Aspekte der Unternehmensarchitektur. Es kann mehrere Varianten für ein Viewpoint Pattern geben.

- **Information Model Pattern:**
  Viewpoint Patterns benötigen bestimmte Daten für die Visualisierung der Unternehmensarchitektur. Das Information Model Pattern enthält die Informationen darüber, welche Daten für welches Viewpoint Pattern benötigt werden.

- **Data Collection Pattern:**
  Die Datenbereitstellung für die Information Model Patterns ist zeitintensiv und fehleranfällig. Data Collection Patterns beschreiben, wie Daten im Unternehmen effizient erhoben werden können. Zusätzlich geben sie an, wie oft bestimmte Daten aktualisiert werden sollten.

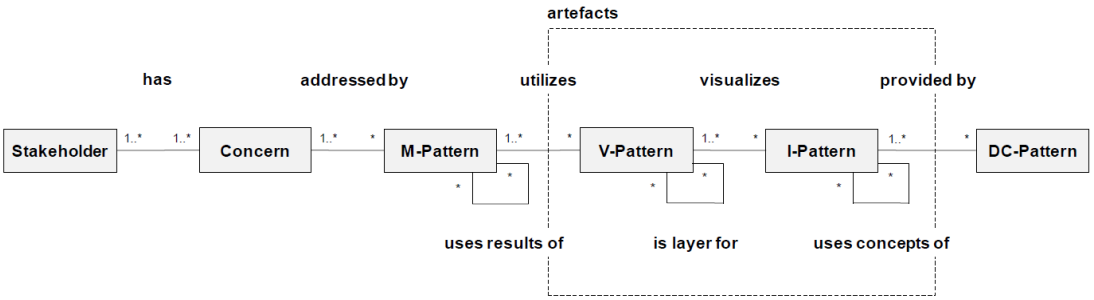

Abbildung 2.4.: Konzeptuelles Diagramm der erweiterten Sprache des EAMPC (Quelle: Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, 2015)

Das darunterliegende konzeptuelle Modell wird in Abbildung 2.4 dargestellt und im Folgenden anhand eines Beispiels erläutert.

Die verschiedenen Elemente werden pro Variante eines Viewpoint Patterns in einem EAM Pattern Graph zusammengefasst. Concerns, Viewpoint Pattern und Information Model Pattern werden in jedem Graphen benötigt, wohingegen Method Pattern,

Stakeholder und Data Collection Pattern optional sind. Im Falle fehlender Method Patterns steht der Concern dabei in direkter Verbindung zum Viewpoint Pattern, was im Graphen durch eine Kante zwischen Concern und Viewpoint dargestellt wird.

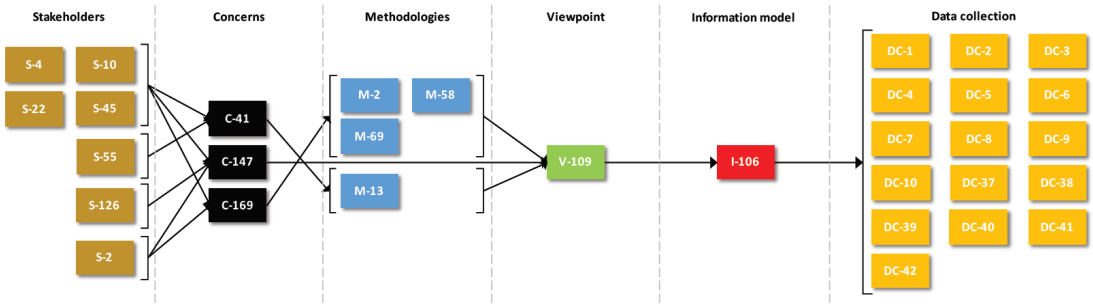

Abbildung 2.5.: Viewpoint V109 Graph (Quelle: Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, 2015)

In Abbildung 2.5 ist der EAM Pattern Graph für Viewpoint V-109 abgebildet. Sieben verschiedene Stakeholder, darunter der CIO[25] (S-4), der Unternehmensarchitekt (S-45) und der IT-Projektmanager (S-2) haben insgesamt drei Concerns[26]:

- Festlegung der verwendeten Infrastruktur für Anwendungen (C-41)
- Zusammenführung von zwei unterschiedlichen Anwendungslandschaften (C-147)
- Architektonische Bewertung von Change Requests (C-169)

Diese Concerns werden in den Methodology Patterns M-2, M-58, M-69 und M-13 behandelt. Methodology M-2 beispielsweise ist die Konformitätsanalyse der Anwendungslandschaft. Dieses Methodology Pattern beschreibt eine Vorgehensweise, mit der überprüft werden kann, ob die Anwendungslandschaft eines Unternehmens den definierten IT-Standards entspricht.[27]

[25] Chief Information Officer
[26] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 120ff.
[27] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 28.

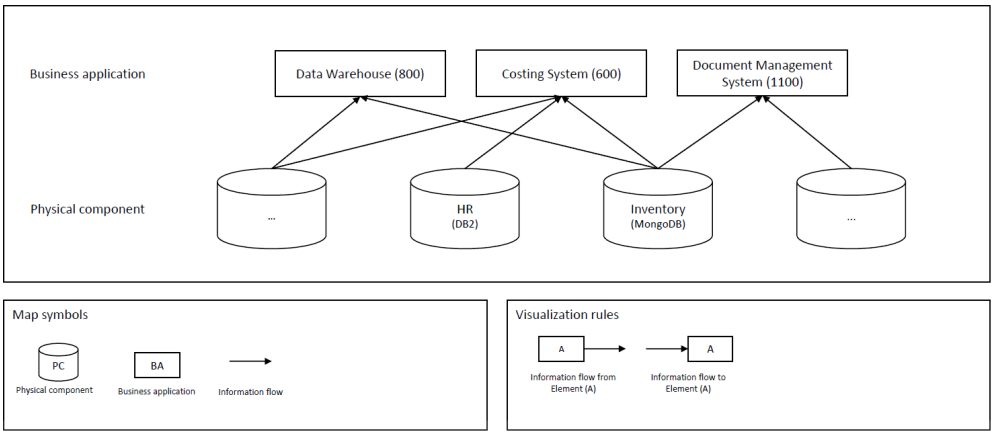

Abbildung 2.6.: Viewpoint V-109 (Quelle: Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, 2015)

Die Methodolgy Patterns verwenden Viewpoint V-109 – „Transparency about used physical components for business applications“, der in Abbildung 2.6 dargestellt wird. Dieses Viewpoint Pattern visualisiert, welche physischen Komponenten von den Business-Anwendungen verwendet werden. Es besteht aus zwei Schichten: den physischen Komponenten und den Business-Anwendungen.[28] Zusätzlich gibt es eine Legende, in der die verwendeten Symbole erläutert werden sowie die Regeln, nach denen diese Visualisierung erstellt wird.

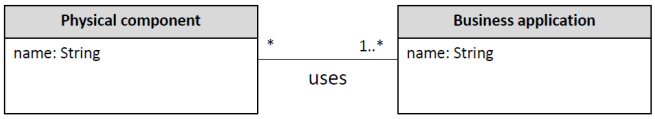

Abbildung 2.7.: Information Model I-106 (Quelle: Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, 2015)

Abbildung 2.7 zeigt das Information Model Pattern I-106, das durch den Viewpoint V-109 visualisiert wird. Information Model Patterns bestehen aus einem fachlichen Datenmodell und einem Glossar-Auszug, in dem die verwendeten Entitäten erklärt werden, hier also die physischen Komponenten und die Business-Anwendungen.[29]

[28] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 51.
[29] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 81.

Die Daten für Information Model I-106 werden durch eine Vielzahl an Data Collection Patterns bereitgestellt. Data Collection Patterns beziehen sich auf genau eine EAM-Klasse und enthalten Informationen über die folgenden Dimensionen[30]:

1. den Verantwortlichen für die Pflege der EAM-Klasse
2. die Datenquelle, aus der die Daten stammen
3. das Format, in dem die Daten vorliegen
4. die Häufigkeit, mit der die Daten aktualisiert werden.

D-1 bis D-10 enthalten Best Practises zur Datenerhebung bezüglich der Business-Anwendungen. Data Collection Pattern D-7 beispielsweise nennt den Unternehmensarchitekten als Verantwortlichen und eine Aktualisierungshäufigkeit nach Bedarf.[31]

[30] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 99.
[31] Vgl. Khosroshahi et al., *Enterprise Architecture Management Pattern Catalog*, S. 126.

3. Anforderungsanalyse

Dieses Kapitel widmet sich den Anforderungen an die Anwendung, die im weiteren Verlauf der Arbeit als Grundlage für den Entwurf dienen. Die Anforderungsanalyse beginnt mit einer Beschreibung der Stakeholder der Anwendung, anschließend werden die Anwendungsfälle vorgestellt und daraus die funktionalen Anforderungen abgeleitet. Es folgt die Beschreibung der nichtfunktionalen Anforderungen.

3.1. Stakeholder

Für die Nutzung der Anwendung wurde der Stakeholder „Anwender“ identifiziert. Seine Rolle und seine Ziele werden in Tabelle 3.1 beschrieben.

Rolle	Anwender
Beschreibung	Der Anwender ist ein Mitarbeiter von Nordantech oder einer Partner-Unternehmensberatung und benutzt die Anwendung. Er ist vertraut mit der Bedienung von Webanwendungen und kennt sich in der fachlichen Domäne aus. Er hat kein spezielles technisches Wissen.
Ziel	gute Bedienbarkeit, schnelles Erlernen, wenig Arbeitsaufwand, Zusammenhänge schnell erkennen

Tabelle 3.1.: Stakeholder: Anwender

3.2. Anwendungsfälle

Um die funktionalen Anforderungen aus der Außensicht zu beschreiben, werden Anwendungsfälle verwendet. Für die Formulierung der Anwendungsfälle dient die

„Vollständig ausgearbeitete Use-Case-Schablone“[1] als Grundlage:

> *„**<Titel>** <der Titel sollte das Ziel in einem kurzen Satz mit aktivem Verb beschreiben>*
> ***Anwenderkontext:** <eine längere Zielbeschreibung, bei Bedarf die normalen Bedingungen seines Auftretens>*
> ***Umfang:** <Design-Umfang, das entstehende System als Blackbox>*
> ***Ebene:** <Auswahl: Überblick, Anwenderziel, Subfunktion>*
> ***Primärakteur:** <ein Rollenname für den Primärakteur oder seine Beschreibung>*
> ***Stakeholder und Interessen:** <Liste der Stakeholder und Schlüsselinteressen im Use Case>*
> ***Vorbedingung:** <der zugrunde gelegte Stand der Dinge>*
> ***Invarianten:** <wie die Interessen bei allen denkbaren Ergebnissen gewahrt werden>*
> ***Nachbedingungen:** <der Stand der Dinge, wenn das Ziel erreicht wird>*
> ***Trigger:** <was den Use Case auslöst, kann ein Zeit-Ereignis sein>*
> ***Standardablauf:***
> *<hier folgen die Schritte des Szenarios vom auslösenden Ereignis bis zur Ausgabe des Ziels und aller anschließenden Datenbereinigungen>*
> *<Schritt #> <Aktionsbeschreibung>*
> ***Erweiterungen:***
> *<hier folgen die Erweiterungen, eine nach der anderen, alle mit einem Bezug zu einem Schritt im **Standardablauf**>*
> *<veränderter Schritt> <Bedingung>: <Aktion oder Teil-Use-Case>*
> *<veränderter Schritt> <Bedingung>: <Aktion oder Teil-Use-Case>*
> ***Liste der Technik- und Datenvariationen:***
> *<hier folgen Varianten, die schließlich zu einer Verzweigung im Szenario führen>*
> *<Schritt oder Variante #> <Variantenliste>*
> *<Schritt oder Variante #> <Variantenliste>*
> ***Verwandte Informationen:** <alle zusätzlichen Informationen, die für Ihr Projekt erforderlich sind“*

[1] Vgl. Cockburn, *Use Cases effektiv erstellen*, S. 151 f.

In dieser Arbeit werden der Anwenderkontext, die Invarianten, die Liste der Technik- und Datenvariationen und die verwandten Informationen aufgrund des Umfangs der Anwendung nicht benötigt und im Folgenden nicht aufgeführt.

Use Case 1: Projekt auswählen

Umfang: Webanwendung
Ebene: Subfunktion
Primärakteur: Anwender
Stakeholder und Interessen:
Anwender – möchte ein Projekt auswählen, um es sich anzusehen oder zu bearbeiten.
Vorbedingung: Keine.
Nachbedingungen: Ein Projekt ist ausgewählt und der Anwender befindet sich auf der Übersichtsseite.
Trigger: Der Anwender möchte sich ein Projekt ansehen oder es bearbeiten.
Standardablauf:

1. Der Anwender öffnet die Anwendung in seinem Browser.
2. Der Anwender wählt das gewünschte Projekt aus.
3. Das System leitet den Anwender zur Übersichtsseite des ausgewählten Projekts weiter.

Erweiterungen:

2a. Das Projekt ist nicht vorhanden:
  2a1. Der Anwender legt das Projekt neu an (siehe Use Case 2).

Use Case 2: Projekt hinzufügen

Umfang: Webanwendung
Ebene: Anwenderziel
Primärakteur: Anwender
Stakeholder und Interessen:
Anwender – möchte ein im System noch nicht existentes Projekt hinzufügen.
Vorbedingung: Keine.
Nachbedingungen: Ein neues Projekt wurde angelegt und der Anwender befindet

sich auf der Übersichtsseite des neuen Projekts.
Trigger: Der Anwender möchte ein neues Projekt hinzufügen.
Standardablauf:

1. Der Anwender öffnet die Anwendung in seinem Browser.
2. Der Anwender wählt „Projekt hinzufügen“.
3. Das System leitet den Anwender zur Projekterstellung weiter.
4. Der Anwender gibt Titel und Beschreibung ein und bestätigt seine Eingaben.
5. Das System leitet den Anwender zur Übersichtsseite des neu erstellten Projekts weiter.

Erweiterungen:

4a. Der Anwender möchte doch kein neues Projekt erstellen.
  4a1. Der Anwender wählt „Abbrechen“.
  4a2. Das System leitet den Anwender zur Projektauswahl weiter.

Die weiteren Use Cases „3: Datenquellen verwalten“, „4: Maßnahmen und Maßnahmeneffekte verwalten“, „5: Kennzahlen und ihre Effekte verwalten“ und „6: Grafische Darstellung ansehen“ sind im Anhang A.1 zu finden.

In Abbildung 3.1 sind alle Anwendungsfälle in einem Use-Case-Diagramm zusammengefasst.

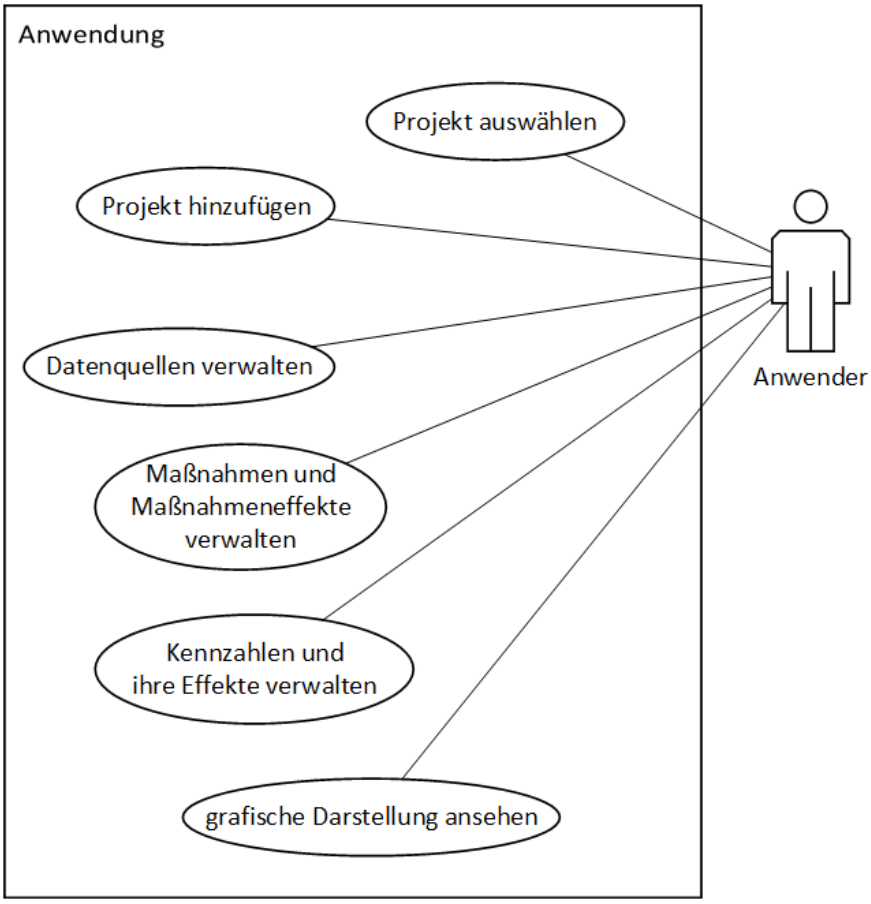

Abbildung 3.1.: Use Case Diagramm (eigene Darstellung)

3.3. Funktionale Anforderungen

Als Vorlage für die Formulierung der funktionalen Anforderungen wird die in Abbildung 3.2 „vollständige Satzschablone ohne Bedingungen“[2] von Klaus Pohl und Chris Rupp verwendet.

[2] Vgl. Rupp und Pohl, *Basiswissen Requirements Engineering – Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*, S. 60.

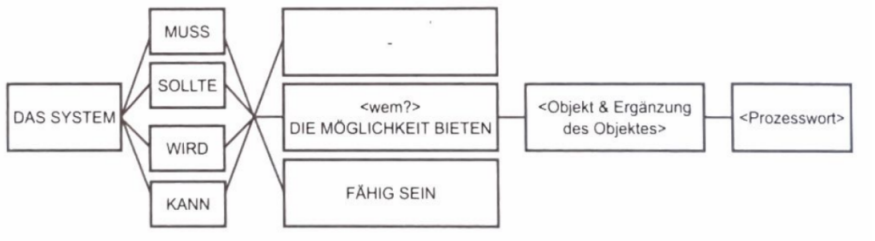

Abbildung 3.2.: Vollständige Satzschablone ohne Bedingungen (Quelle: Rupp und Pohl, *Basiswissen Requirements Engineering – Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*, 2015)

In Tabelle 3.3 werden die funktionalen Anforderungen dargestellt.

Nr.	Use-Case	Anforderung
FA01	1	Das System muss fähig sein, mehrere Projekte zu verwalten.
FA02	1	Das System wird eine Historie aller angelegten Projekte bereitstellen.
FA03	2	Das System muss dem Anwender die Möglichkeit bieten, ein Projekt hinzuzufügen oder zu ändern.
FA04	4	Das System muss dem Anwender die Möglichkeit bieten, in einem bestehenden Projekt Maßnahmen hinzuzufügen, zu ändern und zu löschen.
FA05	5	Das System muss dem Anwender die Möglichkeit bieten, in einem bestehenden Projekt Kennzahlen hinzuzufügen, zu ändern und zu löschen.
FA06	3	Das System muss dem Anwender die Möglichkeit bieten, in einem bestehenden Projekt Datenquellen hinzuzufügen, zu ändern und zu löschen.
FA07	4	Das System muss dem Anwender die Möglichkeit bieten, einen Maßnahmeneffekt hinzuzufügen, zu ändern und zu löschen.

Nr.	Use-Case	Anforderung
FA08	5	Das System muss dem Anwender die Möglichkeit bieten, einen Effekt einer Kennzahl auf eine andere Kennzahl hinzuzufügen, zu ändern und zu löschen.
FA09	6	Das System muss in einer grafischen Übersicht die Zusammenhänge zwischen Datenquellen, Maßnahmen und Kennzahlen darstellen.
FA10	-	Das System sollte dem Anwender die Möglichkeit bieten, zu jeder Zeit das Projekt zu wechseln.

Tabelle 3.3.: Funktionale Anforderungen

3.4. Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen nach Rupp teilen sich auf in folgende Teilbereiche:[3]

- Technologische Anforderungen
- Qualitätsanforderungen
- Anforderungen an die Benutzungsoberfläche
- Anforderungen an sonstige Lieferbestandteile
- Anforderungen an durchzuführende Tätigkeiten
- Rechtlich-vertragliche Anforderungen

Im Rahmen dieser Arbeit werden die Anforderungen an sonstige Lieferbestandteile, die Anforderungen an durchzuführende Tätigkeiten und die rechtlich-vertraglichen Anforderungen nicht betrachtet.

[3] Vgl. Rupp, *Requirements-Engineering und -Management – Aus der Praxis von klassisch bis agil*, S. 268.

Technologische Anforderungen

1. Die Anwendung ist webbasiert und unterstützt folgende Browser:
   - Mozilla Firefox: ab Version 54
   - Google Chrome: ab Version 59
2. Der Server kann lokal betrieben werden.
3. Die Webanwendung ist responsive.
4. Das System muss eine REST-Schnittstelle anbieten.

Qualitätsanforderungen

1. Die Anwendung soll von einem Anwender nach einer Schulung von 10 Minuten fehlerfrei bedient werden können.

Anforderungen an die Benutzungsoberfläche

Die Anforderungen an die Benutzungsoberfläche werden in Form von Skizzen dargestellt. Diese Skizzen dienen der groben Orientierung und der Festlegung der enthaltenen Elemente, weshalb weder die Größe oder die genaue Anordnung der Elemente noch die verwendeten Bezeichnungen umgesetzt werden müssen.

1. Im oberen Bereich gibt es eine Navigationsleiste, die ein schnelles Navigieren zu allen Oberflächen ermöglicht. Sie ist auf allen Oberflächen gleich. Über die Navigationsleiste ist auch ein Wechsel des Projekts möglich.
2. Die Anwendung hat eine Oberfläche „Startseite“, wie in Abbildung 3.3 dargestellt. Sie dient der Auswahl eines Projekts. In dieser Oberfläche enthält die Navigationsleiste keine Elemente.

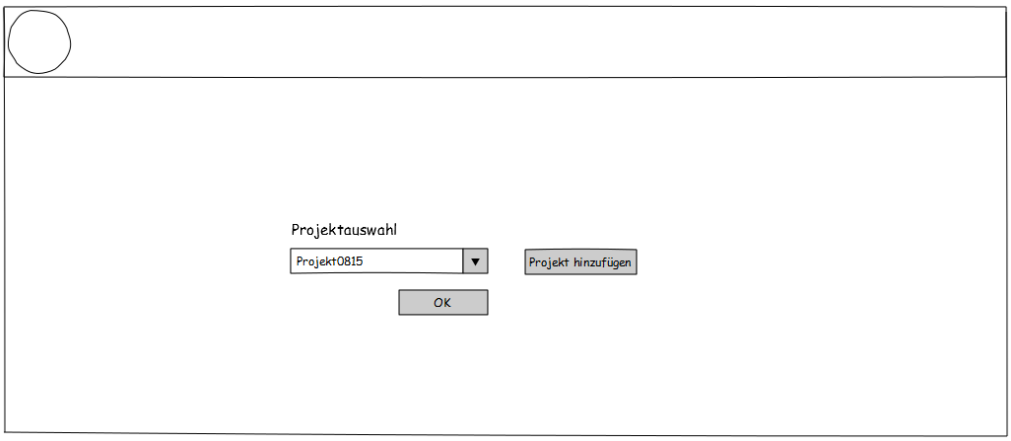

Abbildung 3.3.: Oberflächenentwurf: Startseite (eigene Darstellung)

3. Die Anwendung hat eine Oberfläche „Projekterstellung“, wie in Abbildung 3.4 dargestellt. Diese Oberfläche dient der Erstellung eines Projekts.

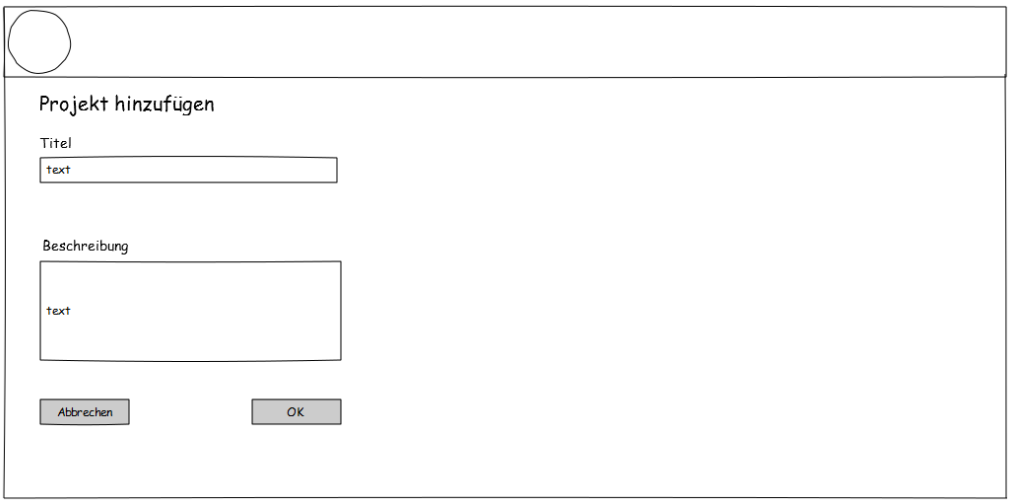

Abbildung 3.4.: Oberflächenentwurf: Projekterstellung (eigene Darstellung)

4. Die Anwendung hat eine Oberfläche „Projektübersicht“, wie in Abbildung 3.5 dargestellt. Die Übersicht zeigt alle bereits hinzugefügten Kennzahlen, Datenquellen und Maßnahmen sowie deren Eigenschaften und Effekte. Zudem können alle Elemente bearbeitet oder gelöscht sowie neue Elemente hinzugefügt werden.

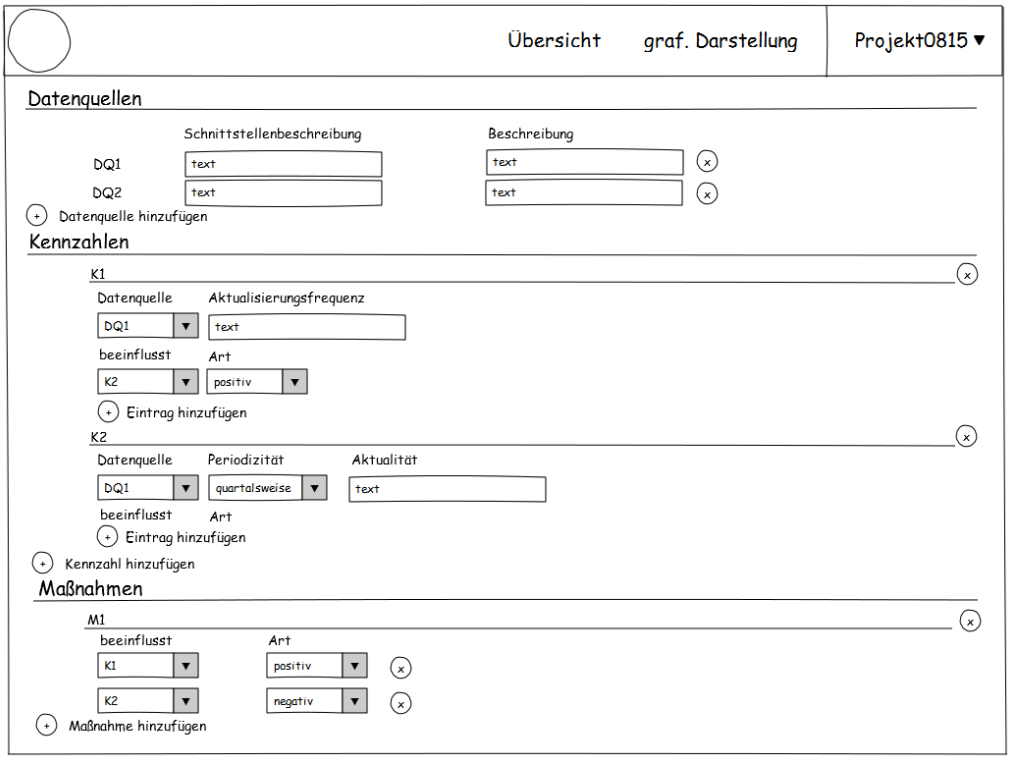

Abbildung 3.5.: Oberflächenentwurf: Projektübersicht (eigene Darstellung)

5. Die Anwendung hat eine Oberfläche „Visualisierung“, wie in Abbildung 3.6 dargestellt. In einer grafischen Darstellung werden die Zusammenhänge zwischen Datenquellen, Kennzahlen und Maßnahmen visualisiert.

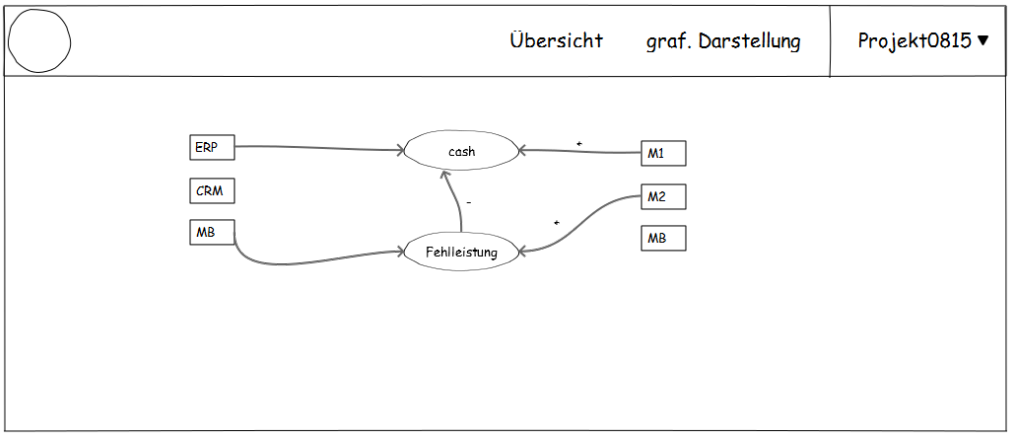

Abbildung 3.6.: Oberflächenentwurf: Visualisierung (eigene Darstellung)

4. Konzeption

In diesem Kapitel wird die Anwendung auf Basis der in Kapitel 3 durchgeführten Anforderungsanalyse konzipiert. In Abschnitt 4.1 werden zunächst auf Basis des in Kapitel 2 vorgestellten EAMPC eigene Concerns sowie ein Viewpoint- und ein Information Model-Pattern für die vorhandene Problemstellung entwickelt. Anschließend werden grundlegende Architekturentscheidungen für die Anwendung getroffen und begründet.

4.1. EAM-Patterns

Eine der grundlegenden Anforderungen an die Anwendung ist eine Visualisierung der Zusammenhänge zwischen Datenquellen, Maßnahmen und Kennzahlen. Diese Visualisierung entsteht in Anlehnung an die Patterns des EAMPC, wie in Kapitel 2.2 beschrieben. Hierfür werden zunächst die Concerns ermittelt und anschließend ein Viewpoint und ein Information Model entwickelt.

Concerns

Die im Vordergrund stehenden Anliegen sind zum einen die Ermittlung der Datenquellen der Kennzahlen, die im Rahmen des Sanierungsmanagements benötigt werden, und zum anderen die Übersicht über die Maßnahmeneffekte:

- **Concern C-E1:** Determine the data sources of KPIs needed for Turnaround Management
- **Concern C-E2:** Get an overview of the effects of measures

Viewpoint Pattern

In Abbildung 4.1 ist der entwickelte Viewpoint V-E1 zu sehen. Die Darstellung besteht aus drei verschiedenen Klassen: Kennzahlen, Datenquellen und Maßnahmen. Jede von ihnen hat eine eigene Farbe, um sie unterscheiden zu können. Die Farbe eines Verbinders ist immer die Farbe der Entität, von der er ausgeht.

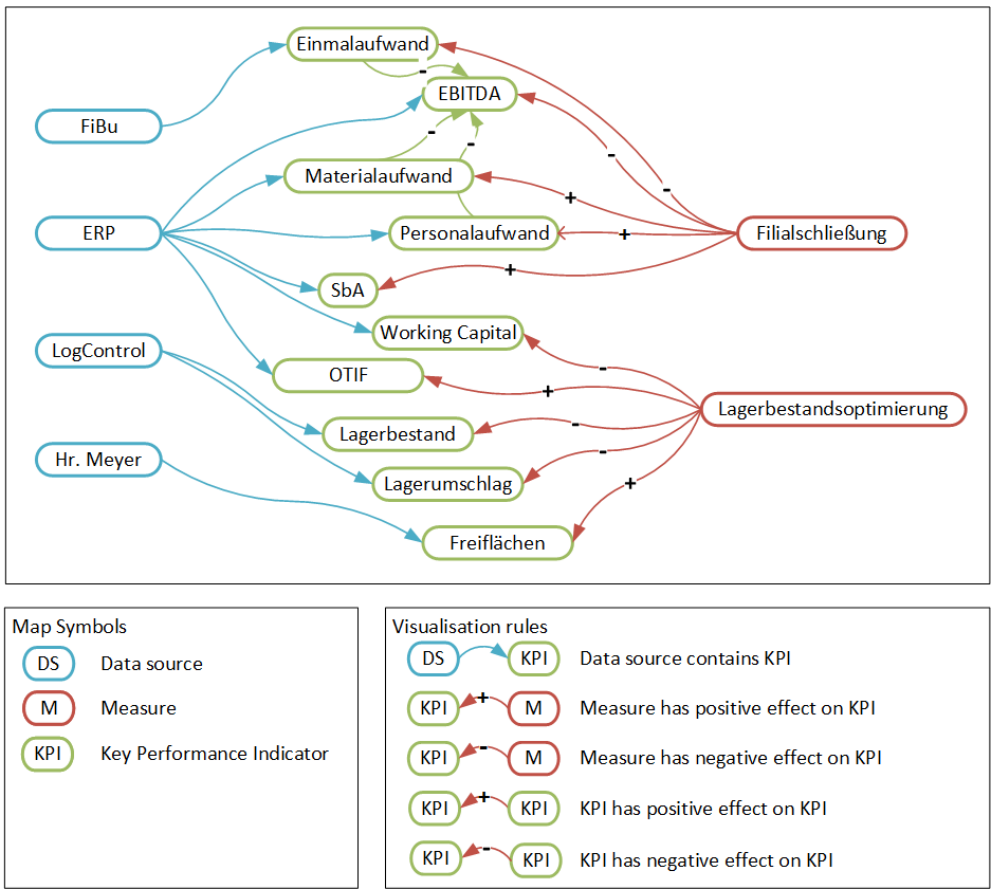

Abbildung 4.1.: Viewpoint V-E1 (eigene Darstellung)

Eine Verbindung zwischen Datenquelle und Kennzahl bedeutet, dass die Kennzahl ihre Datenherkunft in der Datenquelle hat. Diese Verbindung ist nicht beschriftet. Die Verbindung zwischen zwei Kennzahlen bedeutet, dass die eine Kennzahl einen Effekt auf die andere Kennzahl hat. Die Verbindung zwischen Maßnahme und Kennzahl bedeutet, dass die Maßnahme einen Effekt auf die Kennzahl hat.

Die Verbindungen, die einen Effekt repräsentieren, werden mit einem „+“ für einen positiven und mit einem „-“ für einen negativen Effekt gekennzeichnet. Die Richtung der Verbindung geht immer von der beeinflussenden zur beeinflussten Entität.

Abbildung 4.2 zeigt den Zusammenhang zwischen Concerns, Viewpoint und Information Model im Viewpoint Graphen.

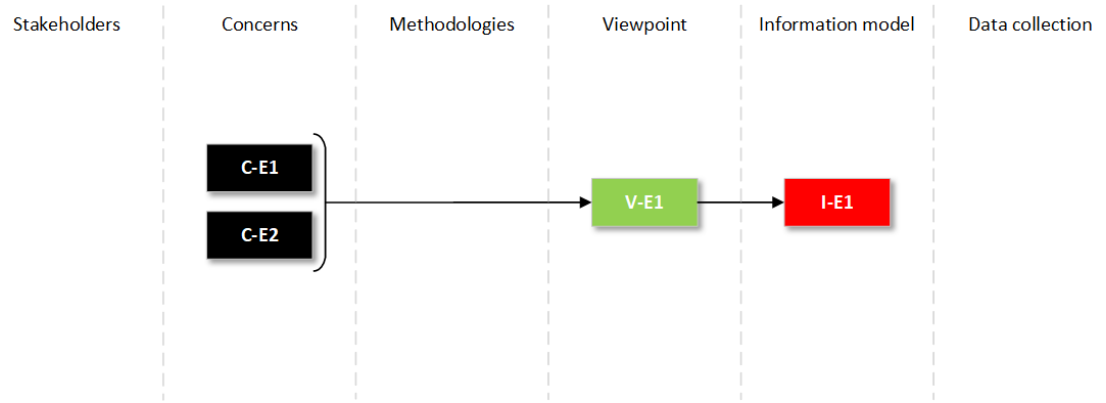

Abbildung 4.2.: Viewpoint V-E1 Graph(eigene Darstellung)

Das vollständige Viewpoint Pattern in englischer Sprache befindet sich im Anhang A.4.

Information Model Pattern

Das Information Model Pattern zeigt alle Beziehungen zwischen den drei Klassen Kennzahlen, Datenquellen und Maßnahmen sowie ihre Attribute.

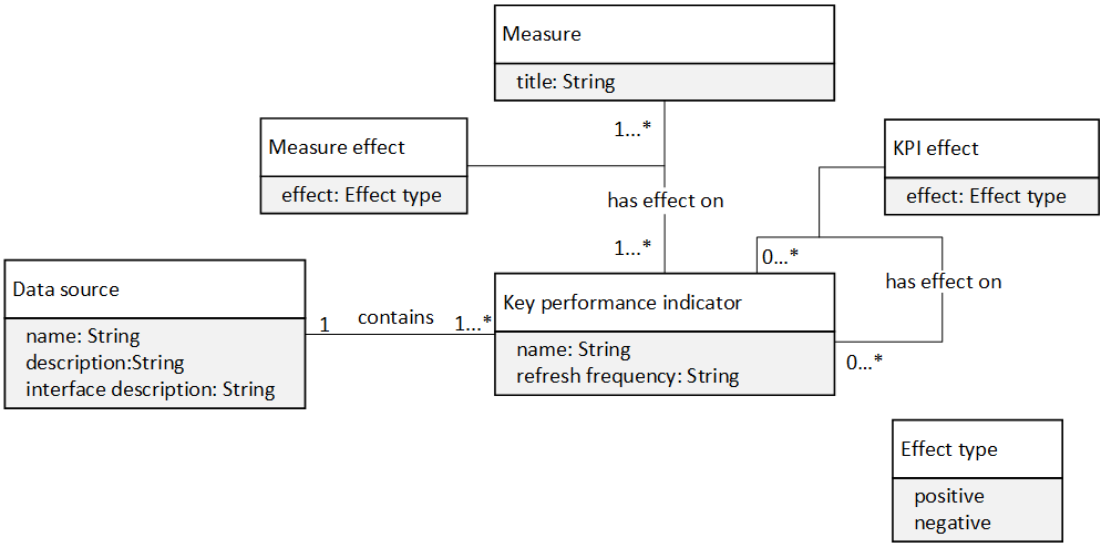

Abbildung 4.3.: Information Model I-E1(eigene Darstellung)

Das vollständige Information Model Pattern in englischer Sprache befindet sich im Anhang A.5.

4.2. Softwarearchitektur

Der Software liegt eine Microservice-Architektur zugrunde: Jeder Service ist eigenständig und hat eine einzige Zuständigkeit. Das gewährt sowohl die Austauschbarkeit der einzelnen Microservices als auch eine einfache Erweiterbarkeit und ermöglicht den Einsatz unterschiedlicher Technologien.[1]

[1] Vgl. Newman, *Microservices: Konzeption und Design*, S. 24ff.

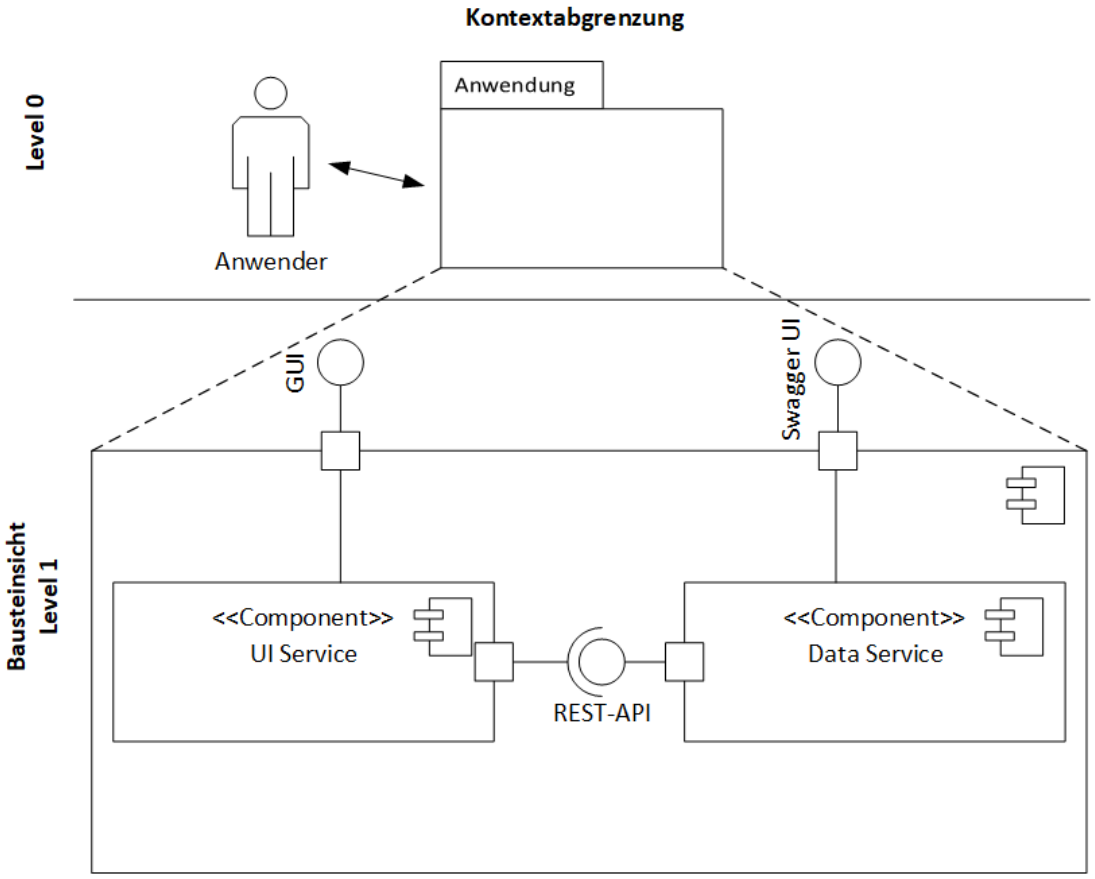

Abbildung 4.4.: Kontextabgrenzung/Bausteinsicht des UI Services (eigene Darstellung)

In Abbildung 4.4 sind die Kontextabgrenzung sowie die Bausteinsicht Level 1 zu sehen. Der Data Service dient als Datenlieferant und speichert Daten in einer Datenbank. Der UI Service stellt dem Anwender die Anwendung zur Verfügung. Die beiden Services kommunizieren über eine REST[2]-Schnittstelle. Eine REST-Architektur zeichnet sich vor allem durch folgende Elemente aus[3]:

- REST impliziert immer eine Client/Server-Architektur. Der Server ist eine Webanwendung, der Client kann ein Webbrowser sein.
- Der Server stellt seine Operationen als Ressourcen zur Verfügung, die über einen einheitlichen Bezeichner, eine URI[4], identifiziert werden.

[2] Representational State Transfer
[3] Vgl. Starke, *Effektive Softwarearchitekturen*, S. 160.
[4] Uniform Resource Identifier

- Die Kommunikation mit dem Client findet via HTTP[5] mithilfe der Standard-Anfragemethoden statt (insbesondere GET, PUT, POST, DELETE).
- Kommuniziert werden Repräsentationen von Ressourcen, beispielsweise im JSON[6]-Format.

REST hat sich „im Webumfeld [...] zur Standardlösung entwickelt“[7], weshalb es viele Tools und Frameworks gibt, die REST unterstützen. Zudem ermöglicht REST aufgrund der statuslosen Kommunikation eine gute Skalierbarkeit und die Kopplung zwischen Client und Server ist sehr gering.

Im Folgenden werden die Services und die Schnittstelle näher beschrieben.

4.2.1. Data Service

Der Data Service stellt die Anwendungsdaten zur Verfügung und speichert sie in einer Datenbank. Für die Umsetzung wurden Node.js und das Framework Express.js ausgewählt, da sie in Kombination viele leistungsfähige Features und Funktionen für Webanwendungen bereitstellen und sich aufgrund der nativen Verarbeitung von JSON-Objekten durch JavaScript für die Entwicklung eines JSON-basierten REST-Services sehr gut eignen.

Node.js ist eine Laufzeitumgebung, mit der JavaScript auch serverseitig eingesetzt werden kann. Zusätzlich bietet Node.js die Möglichkeit, sogenannte Packages einfach einzubinden und zu installieren. Diese Packages bieten zusätzliche Funktionen, wie beispielsweise die einfache Erstellung von Webservern. Große Vorteile von Node.js sind ihre Schnelligkeit und Skalierbarkeit.

Node.js ist im Gegensatz zu herkömmlichen Servern ein Single-Threaded Server. Das bedeutet, dass Node.js statt je eines Threads pro Anfrage nur einen einzigen Haupt-Thread verwendet, der alle Anfragen in eine Anfrage-Queue aufnimmt und abarbeitet. In einer Schleife, der Event-Loop, wird geprüft, ob die nächste Anfrage eine blockierende Ein- oder Ausgabeoperation benötigt. Ist dies der Fall, wird diese Operation von einem Node.js-internen Worker ausgeführt und dabei eine Callback-Funktion

[5] Hypertext Transfer Protocol
[6] JavaScript Object Notation
[7] Vgl. Starke, *Effektive Softwarearchitekturen*, S. 160.

übergeben, die augerufen wird, sobald die Operation durchgeführt wurde. Währenddessen bearbeitet der Haupt-Thread weitere Anfragen, wird also nicht blockiert. So wird verhindert, dass viele Threads erzeugt werden, die Rechenleistung benötigen, obwohl sie lediglich auf das Ergebnis einer Ein- oder Ausgabeoperation warten.[8]

Express.js ist ein Webframework für Node.js, das die Erstellung eines Webservers deutlich vereinfacht. Die zuvor beschriebenen Callback-Funktionen werden im Kontext von Express.js Middleware-Funktionen genannt. Sie realisieren zusätzliche Funktionalitäten, wie beispielsweise das Parsen des Request-Bodies. Jede Middleware-Funktion hat Zugriff auf das Request-Objekt, das Response-Objekt und die jeweils nächste Middleware-Funktion.[9]

Datenbank

Für Daten jeglicher Größe sind Graphdatenbanken der beste Weg, um verbundene Daten zu repräsentieren und abzufragen. Verbundene Daten sind Daten, zu deren Interpretation es erforderlich ist, zu verstehen, wie die einzelnen Elemente zueinander in Beziehung stehen. Oftmals ist es dafür notwendig, diese Verbindungen genau zu bestimmen und zu benennen.[10]

Bei den Daten dieser Anwendung handelt es sich um verbundene Daten und der Fokus liegt vor allem auf den Beziehungen zwischen den Entitäten. Aus diesem Grund wurde eine Graphdatenbank für die Speicherung der Daten gewählt. Auch ist die Struktur der zu entwickelnden Visualisierung die eines Graphen.

Ein weiterer Vorteil ist, dass das Datenmodell jederzeit geändert oder erweitert werden kann, ohne dass die existierenden Daten verändert werden müssen. Sollten also irgendwann in der Zukunft mehr Daten zu den existierenden Entitäten gespeichert werden oder sollte das Modell um neue Entitäten oder Beziehungen erweitert werden, kann das einfach umgesetzt werden, ohne dass die bestehenden Daten und Abfragen angepasst werden müssen.

[8] Vgl. Ackermann, *JavaScript: Das umfassende Handbuch*, S. 817ff.
[9] Vgl. Ackermann, *JavaScript: Das umfassende Handbuch*, S. 842ff.
[10] Vgl. Robinson, Webber und Eifrem, *Graph Databases*, S. xi.

Weiterhin wird das „Labeled Property Graph Model“ als Datenmodell gewählt. Es ist leicht verständlich und zeichnet sich durch folgende Elemente aus[11]:

- Der Graph besteht aus Knoten und Beziehungen.
- Knoten haben Eigenschaften (Key-Value-Paare).
- Knoten können ein oder mehrere Labels haben. Labels gruppieren Knoten.
- Beziehungen sind gerichtet und haben eine Bezeichnung. Sie haben immer einen Start- und einen Endknoten.
- Beziehungen können ebenfalls Eigenschaften haben.

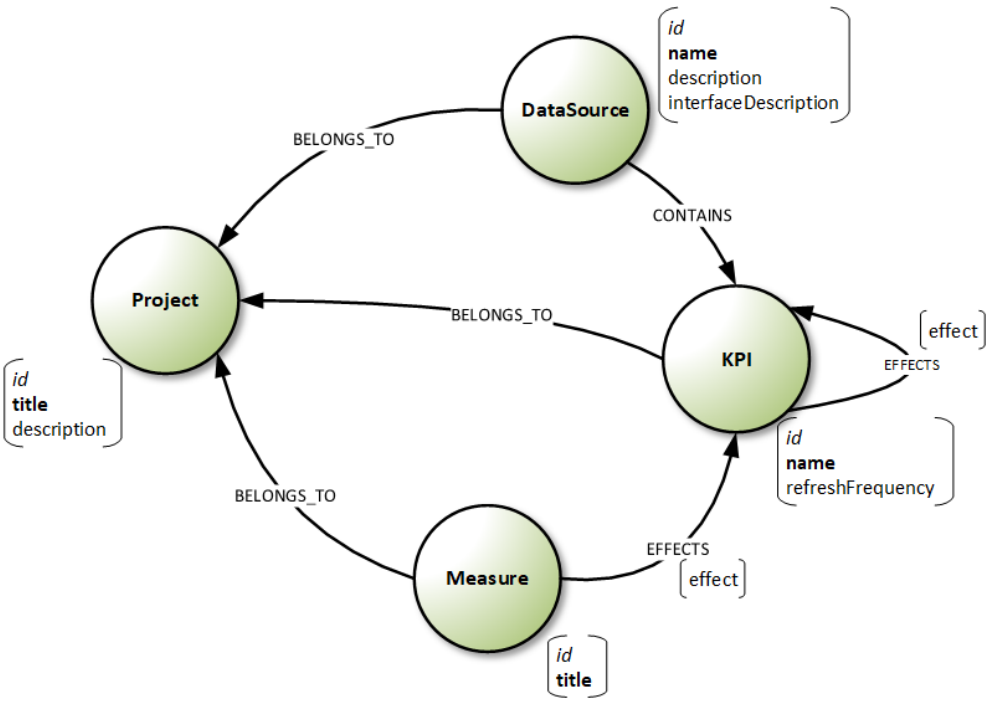

Abbildung 4.5.: Datenmodell (eigene Darstellung)

In Abbildung 4.5 ist das Datenmodell abgebildet. Jeder Knoten ist durch sein Label und eine für dieses Label eindeutige *id* identifizierbar. Diese *id* wird bei Erstellung eines Knotens automatisch hochgezählt. Im Knoten *Project* werden zusätzlich Titel

[11] Vgl. Robinson, Webber und Eifrem, *Graph Databases*, S. 4.

(*title*) und eine Projektbeschreibung (*description*) gespeichert. Der Knoten *DataSource* enthält zusätzlich den Namen der Datenquelle (*name*), eine Beschreibung (*description*) und eine Schnittstellenbeschreibung (*interfaceDescription*), in der angegeben werden kann, ob und in welcher Form eine Schnittstelle zur Datenquelle existiert.

Im Knoten *KPI* werden zusätzlich der Name (*name*) und die Aktualisierungsfrequenz (*refreshFrequency*) der Kennzahl gespeichert. Der Knoten *Measure* enthält zusätzlich den Maßnahmentitel (*title*).

Alle Knoten *DataSource*, *KPI* und *Measure* haben eine Beziehung *BELONGS_TO* zu einem *Project*-Knoten, welche die Zugehörigkeit zu einem Projekt darstellt. *Measure* und *KPI* können zusätzlich eine *EFFECTS*-Beziehung zu einem oder mehreren *KPI*-Knoten haben. Diese Beziehung besitzt die Eigenschaft *effect*, die anzeigt, ob ein Effekt positiv oder negativ ist.

Als Graphdatenbank wird Neo4j[12] verwendet. Neo4j bietet eine SQL-ähnliche Abfragesprache namens Cypher, die einfach zu schreibende, aber mächtige Abfragen ermöglicht. Betreibt man Neo4j im Server-Modus, kann auf die Daten via REST-Schnittstelle zugegriffen werden. Clients können JSON-formatierte Anfragen über HTTP senden und erhalten eine JSON-formatierte Antwort. Die REST-Schnittstelle unterstützt die Ausführung von Cypher-Abfragen. Die Nutzung dieser REST-Schnittstelle erlaubt eine Plattformunabhängigkeit.[13]

4.2.2. UI Service

Der UI Service stellt dem Anwender die grafische Oberfläche zur Verfügung. Er wird als Single Page Application (SPA) entwickelt, was bedeutet, dass die „Oberfläche [...] vollständig in Javascript realisiert und HTML[14] auf dem Client erzeugt [wird]“[15].

In Abbildung 4.6 wird die Arbeitsweise der SPA veranschaulicht: Beim ersten Aufruf des UI Service wird ein HTML-Dokument an den Client ausgeliefert. Zur Laufzeit werden nur noch die Repräsentationen der Ressourcen dynamisch nachgeladen.

[12] https://neo4j.com/
[13] Vgl. Robinson, Webber und Eifrem, *Graph Databases*, S. 77.
[14] Hypertext Markup Language
[15] Vgl. Starke, *Effektive Softwarearchitekturen*, S. 161.

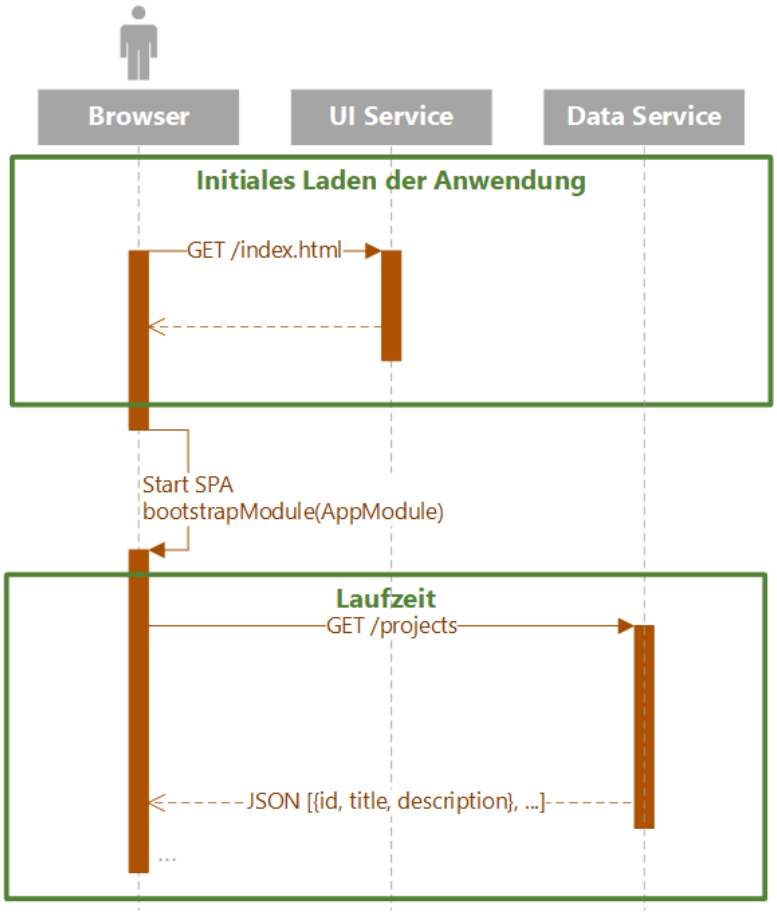

Abbildung 4.6.: Sequenzdiagramm SPA (eigene Darstellung)

Für den UI Service wird Angular verwendet, ein sehr umfassendes JavaScript-Framework. Ausgewählt wurde Angular, weil es Lösungen für viele Aspekte der Anwendungsentwicklung mitbringt, die die Entwicklung einer Single Page Application erleichtern. Entwickelt wird der UI Service in TypeScript. TypeScript ist eine von Microsoft entwickelte Erweiterung des JavaScript-Sprachstandards. Es verfügt über ein starkes Typsystem, wodurch die Entwicklungsumgebung die Programmierung effizient unterstützen kann und Fehler bereits zur Compile-Zeit entdeckt werden. Vor der Auslieferung wird TypeScript immer zu reinem JavaScript transpiliert, wodurch Kompatibilitätsprobleme umgangen werden. Ein weiteres Merkmal ist der Einsatz von

Dekoratoren. Ein Decorator fügt einer Klasse Metainformationen hinzu und ist erkennbar an dem @-Zeichen am Anfang des Namens, z.B. *@Component* oder *@Injectable*.[16]

Angular ist ein komponentenbasiertes Framework, dessen Architektur in Abbildung 4.7 dargestellt wird.

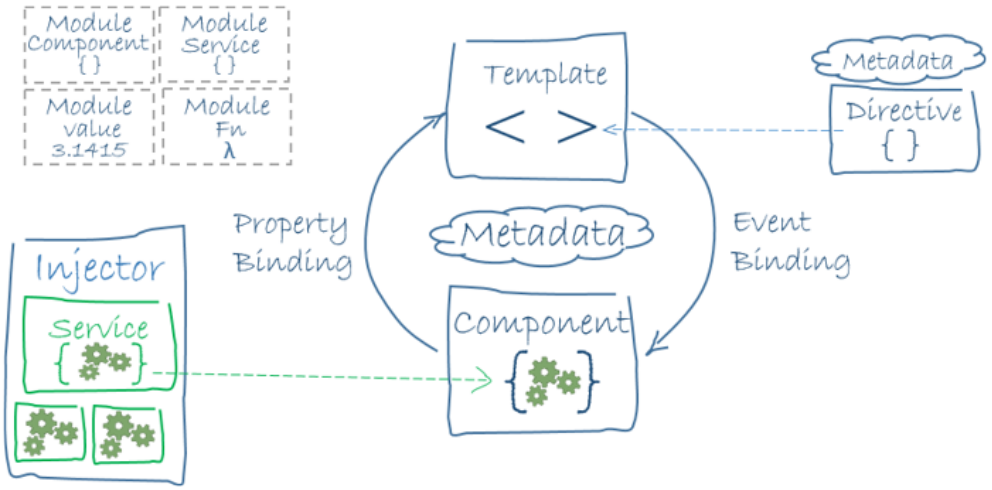

Abbildung 4.7.: Architekturüberblick Angular (Quelle: Google, *Angular Architecture Overview*, 2017)

Der Grundbaustein einer Angular-Anwendung ist die Komponente. Sie beschreibt einen kleinen Teil dieser Anwendung, z.B. ein einzelnes UI-Element und enthält die Anwendungslogik dieses kleinen Teils. Eine Komponente besteht aus einer TypeScript-Klasse, die mit dem Decorator *@Component* beginnt und aus einer View, die durch das zugehörige Template definiert wird.[17] Bei dem Template handelt es sich um eine HTML-Datei, die durch Angular-spezifische Syntax erweitert wird. Die Zuordnung des Templates zur Komponente wird im Decorator definiert. Optional ist zudem eine Style-Datei im CSS[18]-Format, die das Aussehen einer Komponente beeinflusst. Alternativ kann diese Style-Datei auch im SCSS[19]-, SASS[20]- oder LESS-Format vorhanden sein.

[16] Vgl. Woiwode et al., *Angular: Grundlagen, fortgeschrittene Techniken und Best Practises mit TypeScript – ab Angular 4*, S. 27-43.
[17] Vgl. Woiwode et al., *Angular: Grundlagen, fortgeschrittene Techniken und Best Practises mit TypeScript – ab Angular 4*, S. 64.
[18] Cascading Style Sheets
[19] Sassy CSS
[20] Syntactically Awesome Stylesheets

Komponenten können bis in beliebige Tiefe geschachtelt werden. Sie referenzieren sich untereinander und bilden eine Baumstruktur.[21]

```
<!--Property Binding-->
[property] = "value"
<!--Event Binding-->
(event) = "handler"
<!--Two-Way Binding-->
[(ngModel)] = "property"
```

Listing 4.1: Arten von Data Binding in Angular

Angular unterstützt Data Binding, einen Mechanismus, der Teile eines Templates, sogenannte Document-Object-Models (DOM), mit Teilen einer Komponente koordiniert. Ein Binding-Markup im Template zeigt an, auf welche Art die beiden Seiten miteinander verbunden werden sollen. Hier wird in Property Binding, Event Binding und Two-Way Binding unterschieden. Die unterschiedlichen Binding-Markups werden in Listing 4.1 abgebildet. Mit Property Bindings werden Daten von der Komponente an ein DOM-Element übermittelt. Sie werden automatisch aktualisiert, wenn sich die Daten ändern. Mithilfe von Event Bindings kann auf Ereignisse reagiert werden, die im DOM eintreten (z.B. ein Klick auf einen Button). Die Daten fließen also entgegengesetzt der Richtung von Property Bindings. Die Kombination dieser beiden unidirektionalen Verbindungen nennt sich Two-Way Binding: Die Daten aktualisieren sich in beide Richtungen.[22]

Komponenten sind übergeordnet in Modulen organisiert. Diese unterteilen die Komponenten in logische Gruppen und stellen sie nach außen hin zur Verfügung. Module können außerdem Services registrieren. Einstiegspunkt einer Anwendung ist immer das zentrale Root-Modul, das als AppModule bezeichnet wird und mit dem die Anwendung gestartet wird.[23] Ein Angular-Modul ist jedoch nicht vergleichbar mit

[21] Vgl. Woiwode et al., *Angular: Grundlagen, fortgeschrittene Techniken und Best Practises mit TypeScript – ab Angular 4*, S. 88f.
[22] Vgl. Google, *Angular Architecture Overview*.
[23] Vgl. Woiwode et al., *Angular: Grundlagen, fortgeschrittene Techniken und Best Practises mit TypeScript – ab Angular 4*, S. 301f.

einem JavaScript-Modul, sondern ist ein gänzlich anderes Konzept und kann zusätzlich zu den JavaScript-Modulen verwendet werden.[24]

Services sind eigene Klassen, die beispielsweise Business-Logik enthalten können und sich mittels Dependency Injection in Komponenten einbinden lassen. Ein weiteres Beispiel für den Einsatz von Services sind sogenannte Data Services, die eine einheitliche Schnittstelle für Datenzugriffe bieten.[25]

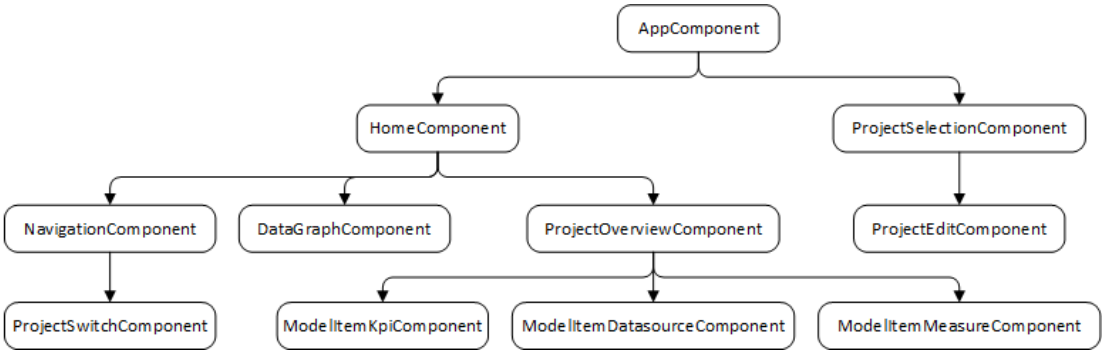

Abbildung 4.8.: Komponentenbaum des UI Services (Quelle: Google, *Angular Architecture Overview*)

Aufgrund des überschaubaren Funktionsumfangs der Anwendung gibt es im UI Service neben dem AppModule nur das RoutingModule, das für die Routen innerhalb der Anwendung zuständig ist.

Die Struktur des UI Service zeigt sich im Komponentenbaum, der in Abbildung 4.8 abgebildet ist. Im Folgenden werden die Funktionen der Komponenten im Einzelnen erläutert:

- **AppComponent:** Die AppComponent wird beim Bootstrapping der Anwendung geladen und bildet den Rahmen, in den alle anderen Komponenten gerendert werden.
- **ProjectSelectionComponent:** Diese Komponente stellt die Projektauswahl dar.

[24] Vgl. Google, *Angular Architecture Overview*.
[25] Vgl. Springer, *Datenmodellierung in Angular: Die Architektur einer Angular-Applikation richtig gestalten.*

- **ProjectEditComponent:** Innerhalb dieser Komponente kann ein neues Projekt erstellt werden.
- **HomeComponent:** Diese Komponente enthält die Inhalte der Anwendung, nachdem ein Projekt ausgewählt wurde. Sie zeigt immer die Navigation und zusätzlich entweder die Projektübersicht oder die Visualisierung.
- **NavigationComponent:** Diese Komponente enthält die Navigationsleiste.
- **ProjectSwitchComponent:** Mit dieser Komponente kann das Projekt gewechselt werden.
- **ProjectOverviewComponent:** Diese Komponente zeigt die Projektübersicht und enthält alle zugehörigen Datenquellen, Kennzahlen und Maßnahmen.
- **ModelItemDatasourceComponent, ModelItemKpiComponent, ModelItemMeasureComponent:** Diese Komponenten zeigen jeweils ein bearbeitbares Objekt ihrer Entität.
- **DataGraphComponent:** Diese Komponente zeigt die Visualisierung der Zusammenhänge zwischen Datenquellen, Kennzahlen und Maßnahmen für das aktive Projekt.

Als notwendige Services wurden die Folgenden identifiziert:

- **ActiveProjectService:** Dieser Service besitzt die Information, welches Projekt ausgewählt wurde und somit gerade aktiv ist.
- **DatasourceService, KpiService, MeasureService, ProjectService:** Diese Services sind Data Services. Sie bilden eine einheitliche Schnittstelle für die Datenzugriffe der jeweiligen Entität und sind zuständig für die Anfragen beim Data Service.
- **ProjectSelectedGuard:** Dieser Service verhindert den direkten Zugriff auf die HomeComponent, wenn kein Projekt ausgewählt wurde, und leitet bei direktem Zugriff an die ProjectSelectionComponent weiter.

Zusätzlich gibt es eine Model-Klasse pro Entität, auf die sowohl Komponenten als auch Services zugreifen können und aus der sich dann Objekte instanziieren lassen.

4.2.3. Schnittstellenbeschreibung

Die REST-Schnittstelle bietet JSON-Objekte an. Hierfür werden Models der Entitäten Projekt, Datenquelle, Kennzahl und Maßnahme modelliert. Abbildung 4.2 zeigt beispielhaft das Projekt-Model. Alle weiteren Models finden sich in Anhang A.2.

```
Project {
    id:                     number
    title:                  string
    description:            string
}
```

Listing 4.2: Model: Project

Eine Liste aller angebotenen Operationen findet sich in Anhang A.3. Beispielhaft werden im Folgenden die von der REST-Schnittstelle angebotenen Operationen vorgestellt, die sich auf Projekte beziehen:

Abfrage aller Projekte

URI:	/projects
Methode:	GET
Parameter:	Keine

Hinzufügen eines Projekts

URI:	/projects
Methode:	POST
Parameter:	• title: Projekttitel • description: Projektbeschreibung

Abfrage eines Projekts

URI:	/projects/{id}
Methode:	GET
Parameter:	• id: Projekt-ID

Ändern eines Projekts

URI:	/projects/{id}
Methode:	PUT
Parameter:	• id: Projekt-ID • title: Projekttitel • description: Projektbeschreibung

Abfrage aller Datenquellen eines Projekts

URI:	/projects/{id}/datasources
Methode:	GET
Parameter:	• id: Projekt-ID

Abfrage aller Kennzahlen eines Projekts

URI:	/projects/{id}/kpis
Methode:	GET
Parameter:	• id: Projekt-ID

Abfrage aller Maßnahmen eines Projekts

URI:	/projects/{id}/measures
Methode:	GET
Parameter:	• id: Projekt-ID

Die Dokumentation der API wird mithilfe einer OpenAPI-Spezifikation[26] realisiert. OpenApi ist ein Beschreibungsformat für REST-APIs, das folgende Informationen enthalten kann:

- Informationen über die verfügbaren Ressourcen (z.B. /projects) und Operationen auf diesen Ressourcen (z.B. GET /projects)
- Operationsparameter für Input und Output jeder Operation
- Authentifizierungsmethoden

[26] https://swagger.io/docs/specification/about/

Swagger UI[27] stellt OpenAPI-Spezifikationen als interaktive API-Dokumentation zur Verfügung. Die Nutzung von OpenAPI in Kombination mit Swagger UI hat den Vorteil, dass der Data Service unabhängig vom UI Service entwickelt und getestet werden kann. Außerdem wird die Schnittstelle ausführlich in einem weit verbreiteten Format dokumentiert. Weiterhin bietet Swagger UI den Nutzern dieser Schnittstelle alle benötigten Informationen grafisch aufbereitet und außerdem die Möglichkeit, alle Operationen auszuprobieren.

Es wird die Version OpenAPI 2.0[28] verwendet, da die neueste Version 3.0 noch nicht von Swagger UI unterstützt wird.

[27] https://github.com/swagger-api/swagger-ui
[28] https://swagger.io/docs/specification/2-0/basic-structure/

5. Implementierung

In diesem Kapitel wird die Implementierung der Anwendung beschrieben. Die Umsetzung basiert auf den in Kapitel 4.2 getroffenen Entwurfsentscheidungen und den ausgewählten Technologien und Frameworks.

5.1. Data Service

Wird eine Anfrage an den Data Service gestellt, wird diese Anfrage zunächst durch alle in der app.js spezifizierten Middleware-Funktionen geleitet. Anschließend prüft Express die Route und ordnet sie einer Weiterleitungsmethode zu. Eine Weiterleitungsmethode wird von einer HTTP-Methode abgeleitet und an eine Instanz der Klasse Express angehängt. Listing 5.1 zeigt die Weiterleitungsmethoden in der app.js für Datenquellen.

```
1 //datasources
2 api.get('/datasources', routes.datasources.list);
3 api.get('/datasources/:id', routes.datasources.findById);
4 api.post('/datasources', routes.datasources.add);
5 api.delete('/datasources/:id', routes.datasources.delete);
6 api.put('/datasources/:id', routes.datasources.update);
```

Listing 5.1: Auszug aus app.js

Alle Weiterleitungsmethoden, in denen eine Anfrage die Ressource Datenquellen anfragt, werden an die Datei routes/datasources.js weitergeleitet. Die Operation GET /datasources beispielsweise wird in Zeile 2 an die Methode *list* in routes/datasources.js weitergeleitet. Diese Methode wird in Listing 5.2 abgebildet.

```
exports.list = function(req, res, next) {
    DataSources.getAll(dbUtils.getSession(req))
        .then(response => writeResponse(res, response))
        .catch(next);
};
```

Listing 5.2: Auszug aus routes/datasources.js

Die Methode *list* stellt die Verbindung zum Model her und ruft dort die Methode *getAll* auf. Das return statement von *getAll* wird dann als Antwort weiterverarbeitet.

In Listing 5.3 ist die Methode *getAll* des Datasource-Models abgebildet. Zurückgegeben wird das in der Methode *_manyDataSources* in ein Array von Datenquellen-Objekten transformierte Ergebnis der Anfrage an die Neo4j-REST-API mit der in den Zeilen 2-8 definierten Cypher-Abfrage.

```
var getAll = function(session) {
    var query = [
        'MATCH (datasource:DataSource)',
        'OPTIONAL MATCH (datasource)
            -[:BELONGS_TO]->(project:Project)',
        'RETURN DISTINCT datasource,',
        'collect(DISTINCT { projectId:project.id}) AS project'
    ].join('\n');

    return session.run(query)
        .then(result => _manyDataSources(result));
};
```

Listing 5.3: Auszug aus models/datasources.js

Anfragen, die die anderen Entitäten betreffen, werden analog in den jeweiligen Routes- und Model-Dateien behandelt.

Die Verbindung zu Neo4j wurde mithilfe des neo4j-drivers[1] realisiert, einem Datenbank-Treiber für Neo4j, der über ein Node.js-Modul eingebunden wird. Er ermöglicht die Anfragen an die Neo4j-REST-API.

[1] https://github.com/neo4j/neo4j-javascript-driver

5.2. UI Service

Für die Implementierung des UI Service wird das Tool Angular CLI[2] verwendet. Es beinhaltet Vorlagen und Befehle für wiederkehrende Aufgaben, wie z.B. das Erstellen eines Grundgerüsts einer Komponente. Angular CLI ist ein Kommandozeilentool auf Basis von Node.js und ist sehr hilfreich, um eine einheitliche Struktur innerhalb des Angular-Projektes beizubehalten. Ein integrierter Webserver präsentiert die Anwendung im Browser. Dieser integrierte Webserver erleichtert das Programmieren, da jede Änderung an einer Datei eine Aktualisierung im Browser auslöst.

Die Entscheidung der Verwendung der Angular CLI wurde getroffen, da sie aufgrund der vorgegebenen Projektstruktur und der Erzeugung von vordefinierten Strukturen wie Komponenten und Services für eine enorme Zeitersparnis sorgt. Zudem ist die Qualität des generierten Codes sehr hoch, weil dieser dem offiziellen Angular Style Guide entspricht.

Zusätzlich wird Angular Material[3] eingesetzt, das Komponenten im Material Design von Google anbietet.

5.2.1. Codebeispiele

Um die Funktionsweise des Codes zu demonstrieren, wird im Folgenden der Datenfluss bezogen auf Datenquellen innerhalb der ProjectOverviewComponent vorgestellt:

```
1 this._projectService
2     .getDatasourcesForSingle(this.initialProject.id)
3     .subscribe(datasources => {
4         this.datasources = datasources;
5     });
```

Listing 5.4: Auszug aus project-overview.component.ts

Listing 5.4 zeigt die Registrierung beim ProjectService bei Initialisierung der ProjectOverviewComponent, um die Datenquellen vom Data Service zu beziehen.

[2] https://cli.angular.io
[3] https://material.angular.io

In Zeile 2 wird die Methode *getDatasourcesForSingle* mit dem Parameter der aktuellen Projekt-ID aufgerufen.

```
1 private apiEndpoint = 'http://localhost:3000/api/projects/';
2 private datasourcesPath = '/datasources';
3
4 [...]
5
6 getDatasourcesForSingle(id: number) {
7     const url = this.apiEndpoint + id + this.datasourcesPath;
8     return this.http
9         .get(url)
10        .map(res => res as Datasource[]);
11 }
```

Listing 5.5: Auszug aus project-service.ts

Listing 5.5 zeigt die Implementierung dieser Methode: Die Variable *apiEndpoint* (Zeile 1) definiert die URL[4], unter der die Projekte vom Data Service abgerufen werden. Die Variable *datasourcesPath* definiert den Pfad der Datenquellen. In Zeile 7 werden sie verwendet, um zusammen mit der *id* die URL zum Abruf der Datenquellen zusammenzusetzen. Zurückgegeben wird ein Observable von Datasources.

Da ein Observable eine unbestimmte Menge an Daten zu einer unbestimmten Zeit liefert, kann es in der ProjectOverviewComponent nicht einfach zugewiesen werden, sondern wird von ihr abonniert und dann zugewiesen (siehe Listing 5.4, Zeilen 3-4).

Die Verwendung von Observables in Services ist ein wichtiger Bestandteil der Prinzipien von Angular.

```
1 <h3>Datenquellen</h3>
2
3 <div *ngIf="datasources && datasources.length > 0"
4      class="model-item appear">
5   <hy-model-item-datasource
```

[4] Uniform Resource Locator

```
6       *ngFor="let datasource of datasources"
7       [item]="datasource"
8       (updatedEvent)="updateDatasources($event)"
9       (removedEvent)="deleteDatasource($event)">
10    </hy-model-item-datasource>
11
12  [...]
13
14  </div>
```

Listing 5.6: Auszug aus project-overview.component.html

Listing 5.6 zeigt einen Auszug aus dem Template der ProjectOverviewComponent. Das **ngIf* in Zeile 3 zeigt eine der Markup-Erweiterungen von Angular und bewirkt, dass der gesamte Codeblock nur dann gerendert wird, wenn die nachfolgende Bedingung erfüllt ist, also wenn Datenquellen existieren.

Der innenliegende *<hy-model-item-datasource>*-Block bindet die Komponente ModelItemDatasourceComponent ein und iteriert dabei über die *datasources*: Jede datasource wird per Property Binding an das *item* gebunden (Zeilen 6-7).

Der Empfang dieses *items* ist in Listing 5.7 zu sehen: Die ModelItemDatasourceComponent erwartet einen *item*-Input und nutzt das Objekt im Template der Komponente.

```
1  @Input() item: Datasource;
```

Listing 5.7: Auszug aus model-item-datasource.component.ts

Abbildung 5.1 ist ein Screenshot aus der Anwendung und zeigt ein gerendertes Datasource-Objekt.

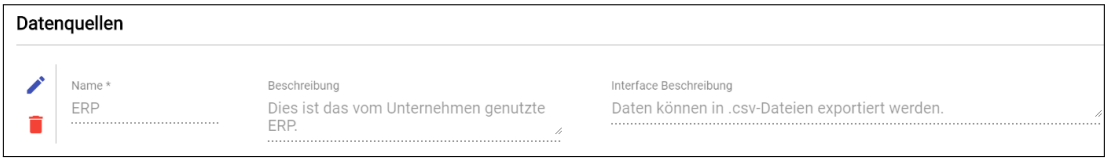

Abbildung 5.1.: Screenshot: Datenquelle

5.2.2. Visualisierung

In der Visualisierungskomponente DataGraphComponent wird die Bibliothek vis.js[5] eingesetzt. Vis.js ist eine dynamische, browserbasierte Visualisierungsbibliothek. Das Network-Modul ermöglicht Visualisierungen von Netzwerken, bestehend aus Knoten und Kanten. Das Modul wurde ausgewählt, da es leicht anzuwenden ist und benutzerdefinierte Formen, Farben und Styles unterstützt.

There are a number of network diagram visualization tools, including those part of the d3.js suite. But perhaps the easiest toolset to get up and running with is the visJS.org package. This Vis.js library makes use of the Canvas api and is optimized for both speed of rendering and level of interactivity.

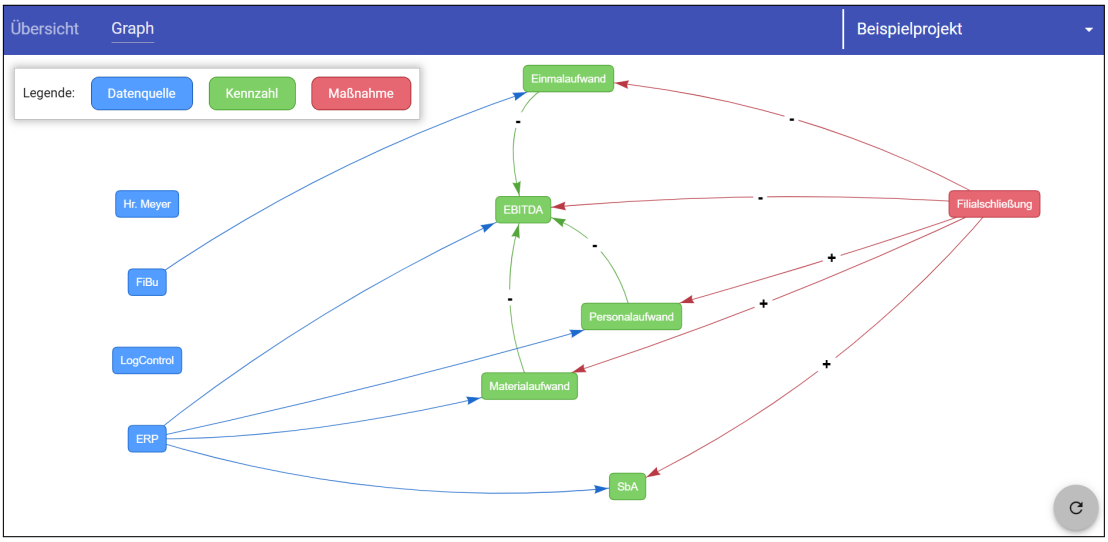

Abbildung 5.2.: Screenshot: Visualisierung

Abbildung 5.2 zeigt einen Screenshot der Anwendung, auf dem die implementierte Visualisierung zu sehen ist. Auf der linken Seite der Darstellung werden die Datenquellen-Knoten abgebildet, auf der rechten Seite die Maßnahmen-Knoten. Um eine Übersichtlichkeit zu garantieren, werden die Datenquellen- und Maßnahmen-Knoten in einem fixen Abstand untereinander positioniert und können nicht verschoben werden. Die Position der Kennzahlen wird dynamisch berechnet und kann per Drag & Drop verändert werden. Das Rendern des Graphen ist nicht idempotent, da

[5] http://visjs.org/

sich die Position der Kennzahl-Knoten bei jeder Neuberechnung verändern kann. Über das Kontextmenü kann die Visualisierung als PNG[6]-Datei gespeichert werden.

5.2.3. Weitere Oberflächen

Im Abschnitt 5.2.2 wurde die Visualisierungsoberfläche bereits ausführlich vorgestellt. Neben der Visualisierung ist auch die Projektübersicht, wie in Abbildung 5.3 abgebildet, ein wichtiger Teil der Anwendung.

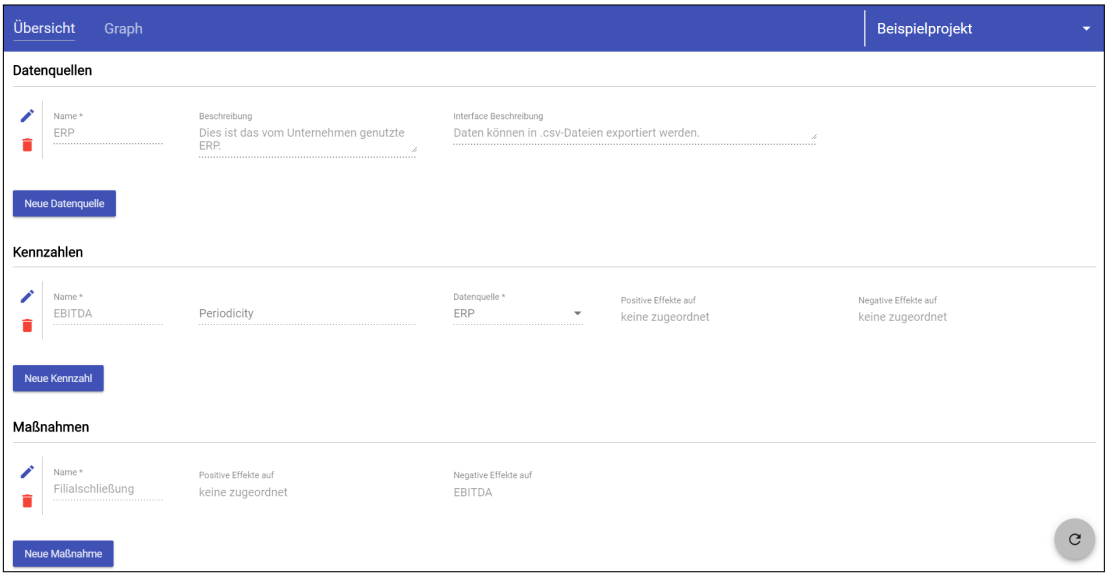

Abbildung 5.3.: Screenshot: Projektübersicht

Hier können Datenquellen, Kennzahlen und Maßnahmen hinzugefügt, geändert oder gelöscht werden. Pflichtfelder sind durch ein * gekennzeichnet und müssen vor dem Speichern ausgefüllt werden.

Die Oberflächen Projektübersicht und Projekterstellung sind in Anhang A.6 abgebildet.

[6] Portable Network Graphics

5.3. REST-Schnittstelle

Das Swagger UI wird mithilfe der Swagger Tools[7] realisiert, die verschiedene Middleware-Funktionen zur Erstellung der Dokumentation bieten. Die API-Spezifikation wurde in YAML[8], einem von Menschen und Maschinen lesbaren Standard zur Datenserialisierung, geschrieben.

```
1  paths:
2      /projects:
3          get:
4            tags:
5              - projects
6            description: Returns all projects
7            summary: Returns all projects
8            produces:
9              - application/json
10           responses:
11             200:
12               description: A list of projects
13               schema:
14                 type: array
15                 items:
16                   \$ref: '#/definitions/Project'
```

Listing 5.8: Auszug aus swagger.yaml: Operation GET /projects

Listing 5.8 zeigt einen Auszug aus der YAML-Datei der API-Spezifikation, in dem die Operation GET /projects beschrieben wird. Zu sehen ist, dass ein Array mit allen Projekten im JSON-Format mit dem Response-Code 200 zurückgegeben wird. Zeile 16 verweist auf eine zentrale Definition der Entität Projekt innerhalb der YAML-Datei. Der Einsatz von Definitionen ist hilfreich, um Wiederholungen zu vermeiden und den Aufwand und die Fehleranfälligkeit bei Änderungen zu reduzieren.

[7] https://github.com/apigee-127/swagger-tools
[8] YAML Ain't Markup Language, yaml.org

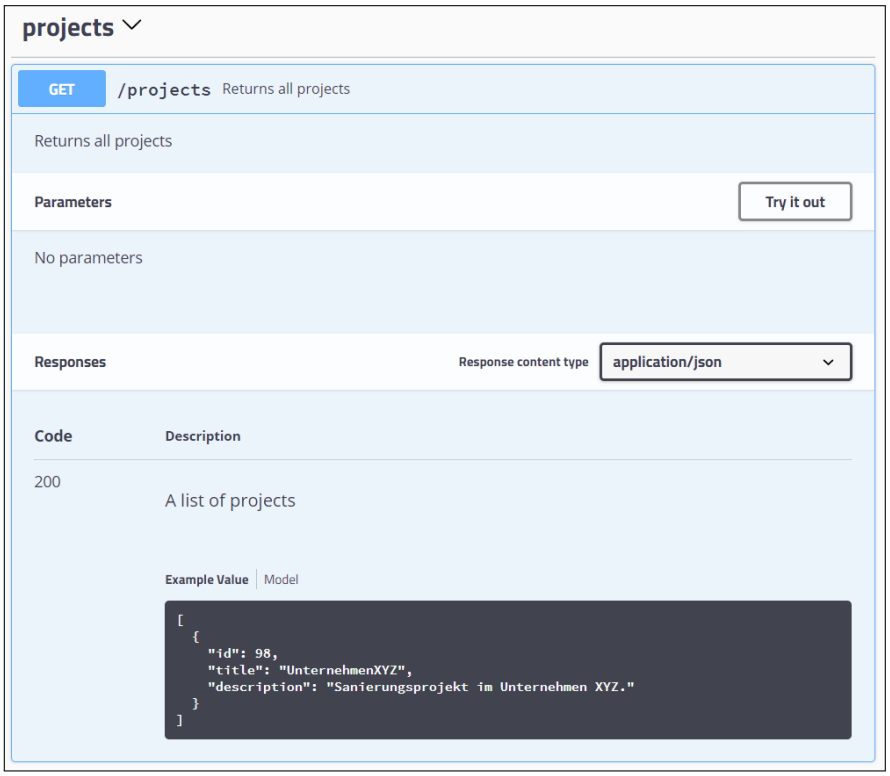

Abbildung 5.4.: Swagger UI: Operation GET /projects

Abbildung 5.4 zeigt die Repräsentation derselben Operation im Swagger UI. Hier ist es auch möglich, über den Button *Try it out* diese Operation auszuprobieren. Alle Operationen werden dabei auf den echten Daten ausgeführt. In diesem Fall werden also alle Projekte zurückgegeben, die in der Datenbank gespeichert sind.

6. Zusammenfassung und Ausblick

Ziel dieser Arbeit war die Entwicklung eines Instruments, das zum einen die Erfassung von Datenquellen, Kennzahlen, Maßnahmen und Effekten ermöglicht und zum anderen eine grafische Übersicht über diese Zusammenhänge bietet. Hierfür wurden zunächst die theoretischen Grundlagen geschaffen und sowohl das Sanierungsmanagement, die fachliche Domäne als auch der EAMPC näher betrachtet.

Die Anforderungsanalyse zeigte die funktionalen und nichtfunktionalen Anforderungen an die Anwendung auf. Use Cases lieferten einen ersten Eindruck davon, welche Funktionen die Anwendung zur Verfügung stellen würde und ausführliche Oberflächenentwürfe ermöglichten einen ersten optischen Eindruck.

Anschließend wurde die zu implementierende Visualisierung auf Basis der EAM-Patterns im EAMPC erstellt. Hierzu wurden zwei Concerns identifiziert und ein Information Model sowie ein Viewpoint für die gegebene Problemstellung entwickelt.

Weiterhin wurden wichtige Entwurfsentscheidungen auf Basis der Anforderungsanalyse getroffen. Für die Anwendung wurde eine Microservice-Architektur mit einem UI Service und einem Data Service mit einer REST-API als Schnittstelle entworfen. Auch die Entscheidung für eine Graphdatenbank und das Datenmodell wurden näher betrachtet. Im nächsten Schritt wurde die Anwendung auf Grundlage der Entwurfsentscheidungen umgesetzt. Die Implementierung wurde anhand von Codebeispielen und Screenshots der Anwendung dokumentiert.

Alle geplanten Funktionen der Anwendung sind voll funktionsfähig. Allerdings wurden die Funktionen nur mit lokalen Servern getestet. Für einen produktiven Einsatz müssten der UI Service und der Data Service auf Servern laufen, die extern zugänglich sind.

Die Umsetzung der Anwendung ermöglicht einen Mehrbenutzer-Betrieb, zu beachten ist jedoch, dass Anomalien bei gleichzeitigen Schreibe- und Leseoperationen anwendungsseitig nicht abgefangen werden. An dieser Stelle müsste eine Sperrstrategie entwickelt werden.

Der Aspekt der Sicherheit wurde aufgrund des Umfangs dieser Arbeit nicht betrachtet. In der aktuellen Implementierung ist die entwickelte REST-API für jeden zugänglich. Hier könnte eine Authentifizierungsprotokoll wie OAuth2[1] eingesetzt werden.

Auch an der Bedienbarkeit der Anwendung könnten einige Punkte verbessert werden: Beim Speichern einer Änderung beispielsweise wird eine Aktualisierung der Daten ausgelöst. Die Anwendung speichert die aktuelle Scrollposition auf der Übersichtsseite nicht und der Browser springt zum Seitenanfang, was Irritationen beim Benutzer hervorrufen kann.

Bei der Erstellung und Änderung von Datenquellen, Kennzahlen und Maßnahmen gibt es zudem relativ viele Speicherschritte. Gibt man viele Dinge auf einmal ein, ist das ständige Speichern eher umständlich und das Data Binding von Angular würde es ermöglichen, diese Speicherschritte zu reduzieren. Ein Usability-Test könnte klären, ob ein signifikanter Anteil an Benutzern dies ebenfalls als störend empfindet und zusätzlich könnten weitere Auffälligkeiten an der Bedienbarkeit der Anwendung aufgedeckt werden.

Die Visualisierung in der Anwendung könnte in mehreren Punkten weiterentwickelt werden: Mit einem Mouse-Over-Effekt könnten sich alle Eigenschaften des jeweiligen Knoten anzeigen lassen. Auch die Positionierung der Knoten könnte gespeichert werden, um einen persistenten Graphen zu gewährleisten. Wenn dann das Verschieben aller Knoten per Drag & Drop ermöglicht wird, kann der Anwender die Visualisierung nach seinen eigenen Wünschen anpassen.

Der bisher beschriebene Ausblick bezieht sich auf die Verbesserung von vorhandenen oder die Einführung von neuen Funktionen. Aber auch der Blick auf die gesammelten Daten lohnt sich: Wird die Anwendung über längere Zeit eingesetzt, könnte man die Datenhistorie auf Gemeinsamkeiten und Muster über verschiedenste Projekte hinweg untersuchen. Ein möglicher Anwendungsfall wäre die Entwicklung von Regeln

[1] https://oauth.net/2/

für Maßnahmeneffekte, also die Identifizierung von Kennzahlen, die immer durch bestimmte Maßnahmen beeinflusst werden (z.B. Maßnahme Lagerbestandsreduktion hat immer einen negativen Effekt auf den Lagerbestand). Für die entwickelte Anwendung könnte dies eine Vorschlagsfunktion ermöglichen, die dem Anwender für eine eingegebene Maßnahme vorschlägt, welche Maßnahmeneffekte diese typischerweise hat.

Auch das konzipierte Pattern-Konstrukt kann weiterentwickelt werden: In der aktuellen Version hat die Beziehung zwischen Kennzahl und Datenquelle im Viewpoint-Pattern keine Beschriftung. Würde diese Beziehung in der Beschriftung Informationen zur Art der Schnittstelle enthalten (automatisch/manuell), könnte der Aufwand für die Datenbeschaffung auf den ersten Blick zumindest grob eingeschätzt werden. Für die Umsetzung könnten die Daten aus der „interfaceDescription“ der Datenquelle entnommen werden.

Auf der Seite der Maßnahmen könnte die Höhe des Netto-Effekts als ein Maß für die Stärke der Verbindungen zwischen Kennzahl und Maßnahme dienen. Eine dünne Linie würde dann einen geringen Effekt symbolisieren, eine dicke Linie einen starken Effekt. Hierzu müsste im Information Model die Beziehung zwischen Maßnahme und Kennzahl um ein Attribut „net effect: Number“ ergänzt werden.

Überlegen könnte man auch eine Erweiterung des Viewpoint-Patterns um die Klasse der Maßnahmenpakete. Diese organisieren die Maßnahmen übergeordnet und könnten eine bessere Übersichtlichkeit bedeuten.

Um einen Überblick über die redundante Speicherung von Kennzahlen im Unternehmen zu erlangen, wäre die Konzeption eines verwandten Patterns vorstellbar, das sich auf Kennzahlen und Datenquellen beschränkt. Die Beziehung zwischen diesen Entitäten könnte dann die Information darüber enthalten, ob eine Datenquelle das führende System für diese Kennzahl ist. Mithilfe dieser Visualisierung könnte zum einen schnell herausgefunden werden, welche Datenquelle das führende System für eine bestimmte Kennzahl ist und zum anderen, in welchen anderen Systemen die Kennzahl zusätzlich gespeichert und verwendet wird. Hierzu müsste die Datenquelle im Information Model um das Attribut „main source: Boolean“ erweitert werden und die Kardinalität zu einer m-n-Beziehung geändert werden.

Als weiteres Anwendungsfeld für das in dieser Arbeit konzipierte Pattern wäre neben dem Sanierungsmanagement der Bereich des Controllings. Auch hier werden Kennzahlen benötigt, um Kontroll- und Steuerungsaufgaben wahrzunehmen. In einem verwandten Pattern könnten die Maßnahmen durch Bereiche des Controllings ersetzt werden, wie beispielsweise Kostenmanagement oder Qualitätsmanagement.

Eine ähnliche Visualisierung wäre auch für die Entwicklung von Big Data-Strategien sinnvoll. So könnte modelliert werden, welche Datenquellen welche Informationen enthalten und welcher Art (z.B. Geodaten) und Qualität (strukturiert/unstrukturiert) diese Daten sind. Auf diese Weise kann eine Übersicht geschaffen werden, die alle zugänglichen Datenbestände visualisiert und insbesondere durch die Schnittstellenbeschreibung die Entwicklung von Big Data-Anwendungen erleichtert. Eine Gruppierung nach Art der Datenquelle (Social Media, interne Datenquellen usw.) wäre vorstellbar.

A. Anhang

A.1. Use Cases

Use Case 3: Datenquellen verwalten

Umfang: Webanwendung
Ebene: Anwenderziel
Primärakteur: Anwender
Stakeholder und Interessen:
Anwender – möchte eine Datenquelle ändern, löschen oder hinzufügen.
Vorbedingung: Der Anwender befindet sich auf der Übersichtsseite eines Projekts.
Nachbedingungen: Der Anwender befindet sich auf der Übersichtsseite des Projekts und seine Änderungen an den Datenquellen wurden gespeichert.
Trigger: Der Anwender möchte eine Datenquelle bearbeiten.
Standardablauf:

1. Wenn der Anwender eine Datenquelle bearbeiten möchte, wählt er den „Stift“ neben der Datenquelle und ändert ein oder mehrere Attribute.
2. Wenn der Anwender eine Datenquelle löschen möchte, wählt er das „X“ neben der Datenquelle und bestätigt, dass er sie wirklich löschen möchte.
3. Wenn der Anwender eine Datenquelle hinzufügen möchte, wählt er „Datenquelle hinzufügen“, das System fügt eine neue Datenquelle hinzu und der Anwender gibt die benötigten Informationen ein.
4. Der Anwender bestätigt seine Eingaben.
5. Das System speichert die Eingaben des Anwenders.

Erweiterungen:

5a. Das Speichern schlägt fehl:

5a1. Das System informiert den Anwender über das fehlgeschlagene Speichern.

Use Case 4: Maßnahmen und Maßnahmeneffekte verwalten

Umfang: Webanwendung
Ebene: Anwenderziel
Primärakteur: Anwender
Stakeholder und Interessen:
Anwender – möchte eine Maßnahme ändern, löschen oder hinzufügen.
Vorbedingung: Der Anwender befindet sich auf der Übersichtsseite eines Projekts.
Nachbedingungen: Der Anwender befindet sich auf der Übersichtsseite des Projekts und seine Änderungen wurden gespeichert.
Trigger: Der Anwender möchte eine Maßnahme bearbeiten.
Standardablauf:

1. Wenn der Anwender eine Maßnahme bearbeiten möchte, wählt er den „Stift“ neben der Maßnahme und ändert ein oder mehrere Attribute.
2. Wenn der Anwender eine Maßnahme löschen möchte, wählt er das „X“ neben der Maßnahme und bestätigt, dass er sie wirklich löschen möchte.
3. Wenn der Anwender eine Maßnahme hinzufügen möchte, wählt er „Maßnahme hinzufügen“, das System fügt eine neue Maßnahme hinzu und der Anwender gibt die benötigten Informationen ein.
4. Der Anwender bestätigt seine Eingaben.
5. Das System speichert die Eingaben des Anwenders.

Erweiterungen:

5a. Das Speichern schlägt fehl:
    5a1. Das System informiert den Anwender über das fehlgeschlagene Speichern.

Use Case 5: Kennzahlen und ihre Effekte verwalten

Umfang: Webanwendung
Ebene: Anwenderziel
Primärakteur: Anwender
Stakeholder und Interessen:

Anwender – möchte eine Kennzahl ändern, löschen oder hinzufügen.
Vorbedingung: Der Anwender befindet sich auf der Übersichtsseite eines Projekts.
Nachbedingungen: Ein Projekt ist ausgewählt und der Anwender befindet sich auf der Übersichtsseite.
Trigger: Anwender möchte Kennzahlen bearbeiten.
Standardablauf:

1. Wenn der Anwender eine Kennzahl bearbeiten möchte, wählt er den „Stift“ neben der Kennzahl und ändert ein oder mehrere Attribute.
2. Wenn der Anwender eine Kennzahl löschen möchte, wählt er das „X“ neben der Kennzahl und bestätigt, dass er sie wirklich löschen möchte.
3. Wenn der Anwender eine Kennzahl hinzufügen möchte, wählt er „Kennzahl hinzufügen“, das System fügt eine neue Kennzahl hinzu und der Anwender gibt die benötigten Informationen ein.
4. Der Anwender bestätigt seine Eingaben.
5. Das System speichert die Eingaben des Anwenders.

Erweiterungen:

5a. Das Speichern schlägt fehl:
    5a1. Das System informiert den Anwender über das fehlgeschlagene Speichern.

Use Case 6: Grafische Darstellung ansehen

Umfang: Webanwendung
Ebene: Anwenderziel
Primärakteur: Anwender
Stakeholder und Interessen:
Anwender – möchte sich die grafische Darstellung ansehen.
Vorbedingung: Der Anwender befindet sich auf der Übersichtsseite eines Projekts.
Nachbedingungen: Der Anwender befindet sich auf der Oberfläche mit der grafischen Darstellung.
Trigger: Der Anwender möchte sich die grafische Darstellung ansehen.
Standardablauf:

1. Der Anwender wählt in der Navigationsleiste „Visualisierung“
2. Das System leitet den Anwender zur grafischen Darstellung weiter.

Erweiterungen: -

A.2. JSON-Models

```
DataSource {
    id:                     number
    projectId:              number
    name:                   string
    description:            string
    interfaceDescription:   string
}
```

Listing A.1: Model: DataSource

```
KPI {
    id:                     number
    projectId:              number
    name:                   string
    refreshFrequency:       string
    datasourceId:           number
    hasPositiveEffectOn:    [number]
    hasNegativeEffectOn:    [number]
}
```

Listing A.2: Model: KPI

```
Measure {
    id:                     number
    projectId:              number
    name:                   string
    hasPositiveEffectOn:    [number]
    hasNegativeEffectOn:    [number]
}
```

Listing A.3: Model: Measure

A.3. Angebotene Operationen der REST-Schnittstelle

Abfrage aller Projekte

URI:	/projects
Methode:	GET
Parameter:	Keine

Hinzufügen eines Projekts

URI:	/projects
Methode:	POST
Parameter:	• title: Projekttitel • description: Projektbeschreibung

Abfrage eines Projekts

URI:	/projects/{id}
Methode:	GET
Parameter:	• id: Projekt-ID

Ändern eines Projekts

URI:	/projects/{id}
Methode:	PUT
Parameter:	• id: Projekt-ID • title: Projekttitel • description: Projektbeschreibung

Abfrage aller Datenquellen eines Projekts

URI:	/projects/{id}/datasources
Methode:	GET
Parameter:	• id: Projekt-ID

Abfrage aller Kennzahlen eines Projekts

URI:	/projects/{id}/kpis
Methode:	GET
Parameter:	• id: Projekt-ID

Abfrage aller Maßnahmen eines Projekts

URI:	/projects/{id}/measures
Methode:	GET
Parameter:	• id: Projekt-ID

Abfrage aller Datenquellen

URI:	/datasources
Methode:	GET
Parameter:	Keine

Hinzufügen einer Datenquelle

URI:	/datasources
Methode:	POST
Parameter:	• projectId: Projekt-ID • name: Name der Datenquelle • description: Datenquellenbeschreibung • interfaceDescription: Schnittstellenbeschreibung

Abfrage einer Datenquelle

URI:	/datasources/{id}
Methode:	GET
Parameter:	• id: Datenquellen-ID

Löschen einer Datenquelle

URI:	/datasources/{id}
Methode:	DELETE
Parameter:	• id: Datenquellen-ID

Ändern einer Datenquelle

URI:	/projects/{id}
Methode:	PUT
Parameter:	• id: Datenquellen-ID • name: Name der Datenquelle • description: Datenquellenbeschreibung • interfaceDescription: Schnittstellenbeschreibung

Abfrage aller Kennzahlen

URI:	/kpis
Methode:	GET
Parameter:	Keine

Hinzufügen einer Kennzahl

URI:	/kpis
Methode:	POST
Parameter:	• projectId: Projekt-ID • name: Name der Kennzahl • refreshFrequency: Aktualisierungsfrequenz • datasourceId: Datenquellen-ID • hasPositiveEffectOn: Array von Kennzahl-IDs der positiv beeinflussten Kennzahlen • hasNegativeEffectOn: Array von Kennzahl-IDs der negativ beeinflussten Kennzahlen

Abfrage einer Kennzahl

URI:	/kpis/{id}
Methode:	GET
Parameter:	• id: Kennzahl-ID

Löschen einer Kennzahl

URI:	/kpis/{id}
Methode:	DELETE
Parameter:	• id: Kennzahl-ID

Ändern einer Kennzahl

URI:	/kpis/{id}
Methode:	PUT
Parameter:	• name: Name der Kennzahl • refreshFrequency: Aktualisierungsfrequenz • datasourceId: Datenquellen-ID • hasPositiveEffectOn: Array von Kennzahl-IDs der positiv beeinflussten Kennzahlen • hasNegativeEffectOn: Array von Kennzahl-IDs der negativ beeinflussten Kennzahlen

Abfrage aller Maßnahmen

URI:	/measures
Methode:	GET
Parameter:	Keine

Hinzufügen einer Maßnahme

URI:	/measures
Methode:	POST
Parameter:	• projectId: Projekt-ID • title: Maßnahmentitel • hasPositiveEffectOn: Array von Kennzahl-IDs der positiv beeinflussten Kennzahlen • hasNegativeEffectOn: Array von Kennzahl-IDs der negativ beeinflussten Kennzahlen

Abfrage einer Maßnahme

URI:	/measures/{id}
Methode:	GET
Parameter:	• id: Maßnahmen-ID

Löschen einer Maßnahme

URI:	/measures/{id}
Methode:	DELETE
Parameter:	• id: Maßnahmen-ID

Ändern einer Maßnahme

URI:	/kpis/{id}
Methode:	PUT
Parameter:	• title: Maßnahmentitel • hasPositiveEffectOn: Array von Kennzahl-IDs der positiv beeinflussten Kennzahlen • hasNegativeEffectOn: Array von Kennzahl-IDs der negativ beeinflussten Kennzahlen

Viewpoint Overview	
Id	V-E1
Name	Map of data sources, KPIs and measures
Summary	This V-Pattern visualizes the effects measures have on KPIs, the effects KPIs have have on each other and what data source the KPIs are from.

A.4. Viewpoint V-E1

This Viewpoint pattern consists of three classes: Measures are illustrated at the right side of the diagram. The KPIs that they have an effect on are illustrated in the middle. The data sources that contain those KPIs are illustrated at the left side of the diagram. Relationships between data sources and KPIs show that a data source contains the KPI. Relationships between KPIs show that one KPI has an effect on another. Relationships between KPIs and measures show that the measure has an effect on the KPI. The relationships that represent an effect are marked with a „+“ if it is a positive effect and are marked with a „-“ when it is a negative effect. Relationships always have the color of the entity they originate from.

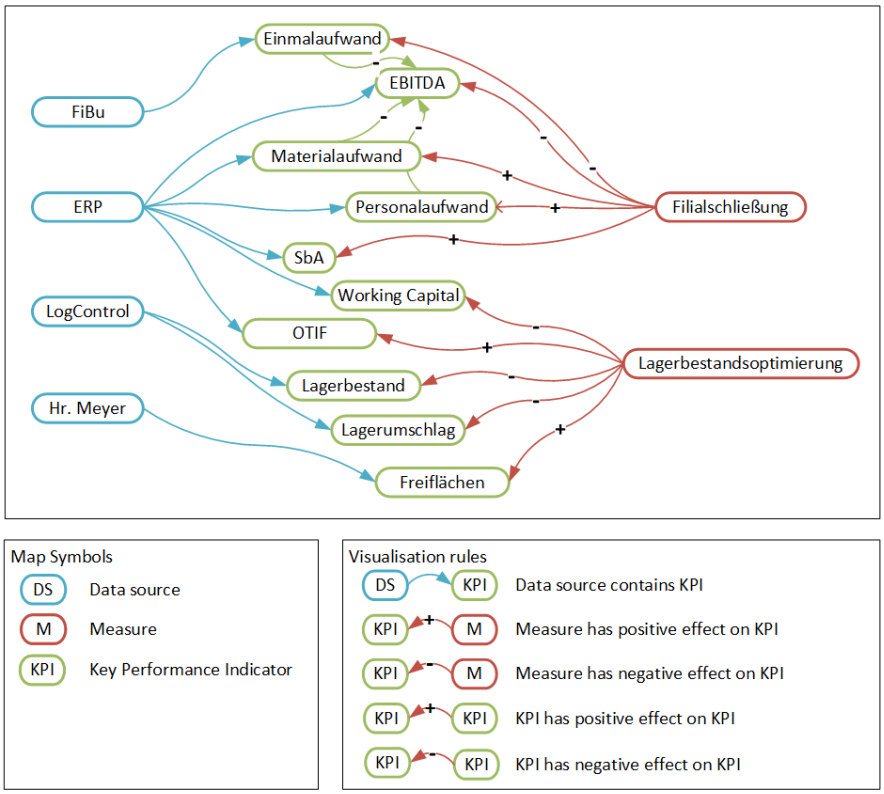

Abbildung A.1.: Viewpoint V-E1 (eigene Darstellung)

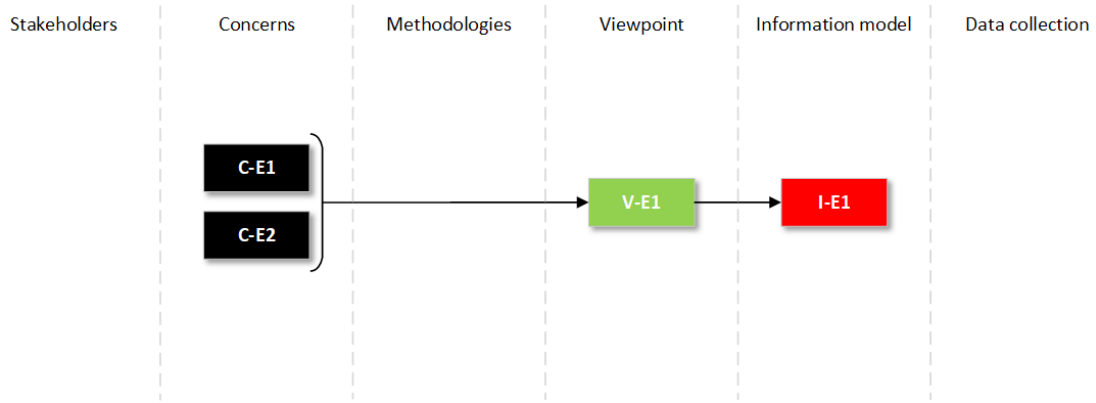

Abbildung A.2.: Viewpoint V-E1 graph (eigene Darstellung)

A.5. Information Model I-E1

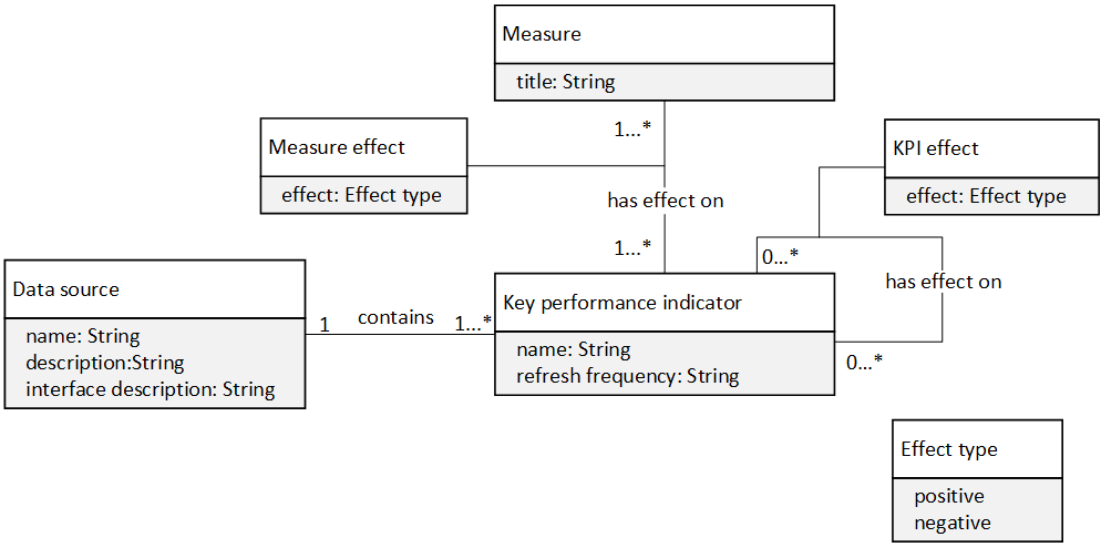

Abbildung A.3.: Information Model I-E1 (eigene Darstellung)

Data source: A data source might be a person, a spreadsheet, a document, a presentation or an information system. An information system is a sociotechnical system that consists of human and machine components with the goal of providing information.[1]

Key Performance Indicator: A Key Performance Indicator is a benchmark that is deliberatly compressed to absolute or relative numbers in order to report in a concentrated form about a numerical ascertainable set of facts.[2]

Measure: A measure is a concrete activity that is necessary to realize a strategy and to achieve business goals in the context of Turnaround Management.[3]

Measure effect: Indicates, which effect a Measure has on a specific KPI.

KPI effect: Indicates, which effect a KPI has on another specific KPI.

[1] Vgl. Abts und Mülder, *Grundkurs Wirtschaftsinformatik: Eine kompakte und praxisorientierte Einführung*, S. 15.
[2] Vgl. Gladen, *Performance Measurement: Controlling mit Kennzahlen*, S. 9.
[3] Vgl. Doppler und Lauterburg, *Change Management – Den Unternehmenswandel gestalten*, S. 209.

Effect type: Different kinds of effect on a KPI: it has to be either positive or negative.

A.6. Screenshots der Anwendung

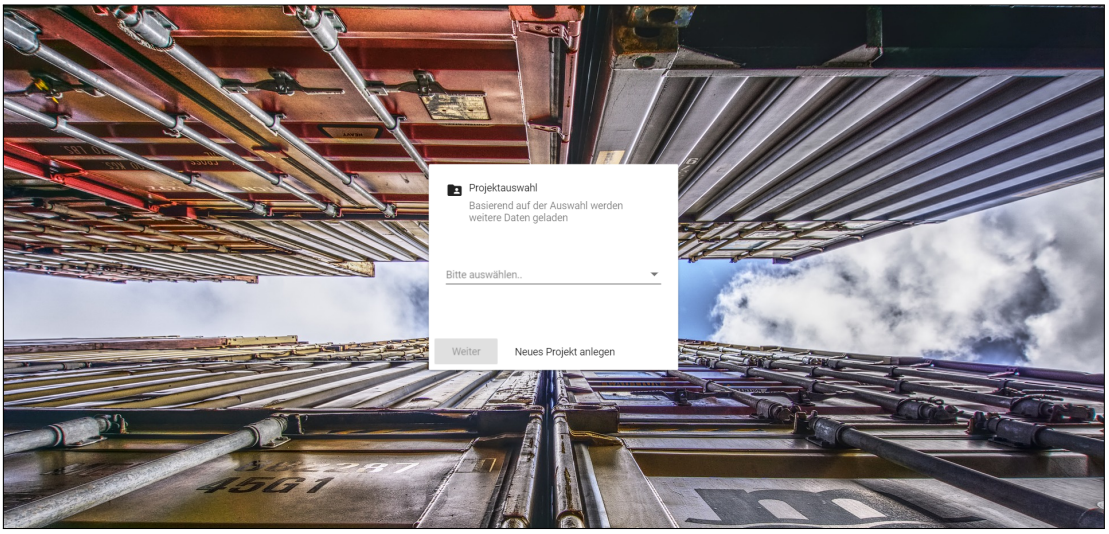

Abbildung A.4.: Screenshot: Projektauswahl

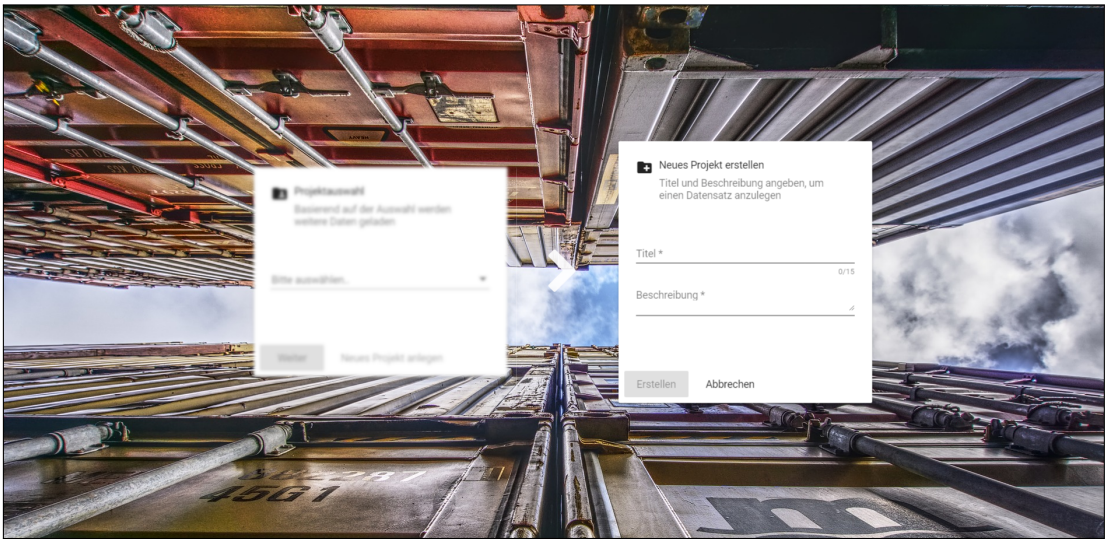

Abbildung A.5.: Screenshot: Projekterstellung

Literatur

Abts, Dietmar und Wilhelm Mülder. *Grundkurs Wirtschaftsinformatik: Eine kompakte und praxisorientierte Einführung*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017.

Ackermann, Philip. *JavaScript: Das umfassende Handbuch*. Bonn: Rheinwerk Verlag, 2016.

Cockburn, Alistair. *Use Cases effektiv erstellen*. 1. Aufl., ND. Frechen: mitp, 2008.

Crone, Andreas. „Die Unternehmenskrise“. In: *Modernes Sanierungsmanagement – Sanierungskonzepte, Finanzierungsinstrumente, Insolvenzverfahren, Haftungsrisiken, Arbeitsrecht und Verhandlungsführung*. Hrsg. von Andreas Crone und Henning Werner. 4. Aufl. München: Vahlen, 2014. Kap. 1.

– „Erstellung von Sanierungskonzepten nach IDW S 6“. In: *Modernes Sanierungsmanagement – Sanierungskonzepte, Finanzierungsinstrumente, Insolvenzverfahren, Haftungsrisiken, Arbeitsrecht und Verhandlungsführung*. Hrsg. von Andreas Crone und Henning Werner. 4. Aufl. München: Vahlen, 2014. Kap. 4.

Crone, Andreas und Henning Werner. „Rechtliche Rahmenbedingungen und Prüfung der Insolvenztatbestände“. In: *Modernes Sanierungsmanagement – Sanierungskonzepte, Finanzierungsinstrumente, Insolvenzverfahren, Haftungsrisiken, Arbeitsrecht und Verhandlungsführung*. Hrsg. von Andreas Crone und Henning Werner. 4. Aufl. München: Vahlen, 2014. Kap. 2.

Doppler, Klaus und Christoph Lauterburg. *Change Management – Den Unternehmenswandel gestalten*. 13. Aufl. Frankfurt am Main: Campus Verlag, 2014.

Eilmann, Sonja et al. „Interessengruppen/Interessierte Parteien“. In: *Kompetenzbasiertes Projektmanagement* 1 (2011), S. 67–97.

Gladen, Werner. *Performance Measurement: Controlling mit Kennzahlen*. Wiesbaden: Springer-Verlag, 2014.

Google. *Angular Architecture Overview*. 2017. URL: `https://angular.io/guide/architecture` (besucht am 18. 12. 2017).

Groß, Paul J. „Erkennen und Bewältigen von Unternehmensschieflagen“. In: *WPg-Sonderheft* (2003).

Hanschke, Inge. *Enterprise Architecture Management – einfach und effektiv*. München: Carl Hanser Verlag, 2012.

Hettich, Christof, Raoul Kreide und Andreas Crone. „Finanzwirtschaftliche Sanierungsmaßnahmen“. In: *Modernes Sanierungsmanagement – Sanierungskonzepte, Finanzierungsinstrumente, Insolvenzverfahren, Haftungsrisiken, Arbeitsrecht und Verhandlungsführung*. Hrsg. von Andreas Crone und Henning Werner. 4. Aufl. München: Vahlen, 2014. Kap. 7.

Hohberger, Stefan und Hellmut Damlachi. *Praxishandbuch Sanierung im Mittelstand*. 3. Aufl. Wiesbaden: Springer-Verlag, 2014.

Institut der Wirtschaftsprüfer e.V. *Anforderungen an die Erstellung von Sanierungskonzepten (IDW S 6), IDW Fachnachrichten 12/2012*. Düsseldorf, 2012.

– *WP Handbuch 2008: Wirtschaftsprüfung, Rechnungslegung, Beratung, Band II*. Düsseldorf, 2008.

Khosroshahi, Pouya Aleatrati et al. *Enterprise Architecture Management Pattern Catalog*. Technische Universität München, sebis, 2015.

Losbichler, H., C. Eisl und C. Engelbrechtsmüller. *Handbuch der betriebswirtschaftlichen Kennzahlen: Key Performance Indicators für die erfolgreiche Steuerung von Unternehmen*. Linde Lehrbuch. Wien: Linde Verlag, 2015.

Newman, Sam. *Microservices: Konzeption und Design*. Frechen: mitp, 2015.

Robinson, Ian, Jim Webber und Emil Eifrem. *Graph Databases*. 2. Aufl. Sebastopol: O'Reilly Media, 2015.

Roth, Sascha et al. „Enterprise Architecture Documentation: Current Practices and Future Directions". In: *Wirtschaftsinformatik*. 2013.

Rupp, Chris. *Requirements-Engineering und -Management – Aus der Praxis von klassisch bis agil*. 6., aktualisierte und erw. Aufl. München: Hanser Fachbuchverlag, 2014.

Rupp, Chris und Klaus Pohl. *Basiswissen Requirements Engineering – Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. 4., überarb. Aufl. Heidelberg: dpunkt.verlag, 2015.

Springer, Sebastian. *Datenmodellierung in Angular: Die Architektur einer Angular-Applikation richtig gestalten*. Sep. 2017. URL: `https://t3n.de/magazin/datenmodellierung-angular-architektur-angluar-application-242317/` (besucht am 18. 12. 2017).

Starke, Dr. Gernot. *Effektive Softwarearchitekturen*. München: Carl Hanser Verlag, 2014.

Werner, Henning. „Leistungswirtschaftliche Sanierungsmaßnahmen". In: *Modernes Sanierungsmanagement – Sanierungskonzepte, Finanzierungsinstrumente, Insolvenzverfahren, Haftungsrisiken, Arbeitsrecht und Verhandlungsführung*. Hrsg. von Andreas Crone und Henning Werner. 4. Aufl. München: Vahlen, 2014. Kap. 6.

Woiwode, Gregor et al. *Angular: Grundlagen, fortgeschrittene Techniken und Best Practises mit TypeScript – ab Angular 4*. Heidelberg: dpunkt.verlag, 2017.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 18. Januar 2018 Katharina Ellermann