ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Dynamic Visualization of Additional Information in Process Models

Masterarbeit an der Universität Ulm

Vorgelegt von:
Florian Gallik
florian-1.gallik@uni-ulm.de
788744

Gutachter:
Prof. Dr. Manfred Reichert
Prof. Dr. Rüdiger Pryss

Betreuer:
Michael Winter

2021

Fassung 11. März 2021

Satz: PDF-LaTeX 2ε

Kurzfassung

Geschäftsprozesse werden mittels Modellierungssprachen wie Business Process Model and Notation (BPMN) 2.0 abgebildet. Meist werden diese nach der Modellierung exportiert und sind danach statisch verfügbar, beispielsweise als Vektorgrafik. Diese Prozessmodelle finden unter anderem Verwendung um Mitarbeiter/-innen die Arbeitsabläufe zu vermitteln. Die Sicht ist für alle Anwender/-innen gleich und bietet demnach auch den gleichen Informationsgehalt, was abhängig von der Abstraktionsebene sehr detailreich sein kann. Je nachdem, wie lange eine Person bereits in einem Unternehmen arbeitet oder wie gut sie sich mit Prozessmodellen auskennt, benötigt diese unterschiedlich viel Informationen. Existierende Systeme bieten keine Funktion an um die Menge an Informationen während der Betrachtung zu regulieren. Um verschiedene Perspektiven zu unterstützen, soll eine Anwendung erstellt werden, die das dynamische Wechseln zwischen Detaillierungsgraden möglich macht.

In dieser Arbeit wird eine Single-Page-Webanwendung erstellt, mit der bereits modellierte BPMN 2.0 Prozessmodelle per Extensible Markup Language (XML) importiert und angezeigt werden. Die Implementierung erfolgt mit der JavaScript Bibliothek React. Zur Anzeige und Manipulation der Modelle wird das Framework Bpmn.io verwendet. Mit der realisierten Webanwendung sollen Benutzer den Detailgrad des Prozessmodells dynamisch während der Betrachtung ändern können. Das bedeutet, dass der Benutzer die Menge an Informationen selbst regulieren und je nach Bedarf an sich anpassen kann. Dies geschieht, indem die Sichtbarkeit einzelner Komponentengruppen wie Annotationen einstellbar ist und Informationen per Overlay hinzufügbar sind. Des Weiteren ist es möglich, Komponenten farblich zu markieren. Die Anwendung, die während dieser Arbeit vorgestellt wird, zeigt auf, dass selbst bei statischen Prozessmodellen der Detailgrad an Informationen dynamisch während der Betrachtung anpassbar ist. Das wird ermöglicht, ohne das Ursprungsmodell permanent zu verändern und ohne dass das Modell mehrfach in verschiedenen Detailabstufungen modelliert und vorhanden ist.

Danksagung

An dieser Stelle möchte ich mich ganz herzlich bei Michael Winter für das interessante Thema und die herausragende Betreuung während dieser Arbeit bedanken.

Meinem Kommilitone Yusuf Kirikkayis gilt ein spezieller Dank für die hilfreichen Anregungen während der Erstellung und das Korrekturlesen dieser Thesis.

Darüber hinaus bedanke ich mich bei Prof. Dr. Manfred Reichert und Prof. Dr. Rüdiger Pryss für die Begutachtung dieser Arbeit.

Nicht zuletzt geht ein besonderer Dank an meine Familie, Freunde und Lebensgefährtin Sarah Schönknecht, die mich über die komplette Studienzeit ermutigt, unterstützt und begleitet haben.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation 1
1.2 Problemstellung 2
1.3 Zielsetzung 3
1.4 Gliederung der Arbeit 3

2 Grundlagen **4**

2.1 BPMN 2.0 4
2.1.1 Diagrammtypen 5
2.1.2 Basis Modellierungselemente 6
2.1.3 Prozessmodell Beispiel 14
2.2 Node.js 16
2.2.1 Npm 16
2.2.2 Npx 16
2.3 Bpmn.io 17
2.3.1 Hauptkomponenten 17
2.3.2 Bpmn-js 18
2.4 Sass 20
2.5 Bootstrap 21
2.6 React 22
2.6.1 Components 22
2.6.2 Hooks 24
2.7 Usability 25

3 Verwandte Arbeiten **28**

4 Anforderungsanalyse **30**

4.1 Funktionale Anforderungen 31

4.2 Nicht-funktionale Anforderungen 35

5 Konzeption **38**
5.1 Architektur 38
5.2 Komponentenbeschreibung 39
5.2.1 Server 39
5.2.2 Webanwendung 39
5.3 Grundgerüst 41
5.4 Mockups 43
5.4.1 Startseite 43
5.4.2 Hauptbildschirm 46
5.4.3 Zusammenspiel 50

6 Implementierung **52**
6.1 Datei hochladen 52
6.2 Initialisierung und BPMN Element Selektion 55
6.3 Auflistung und Hervorhebung der Swimlanes 58
6.4 Overlays 60
6.5 Sichtbarkeit der BPMN 2.0 Elemente 63

7 Anforderungsabgleich **71**
7.1 Funktionale Anforderungen 71
7.2 Nicht-funktionale Anforderungen 72

8 Zusammenfassung und Ausblick **74**
8.1 Zusammenfassung 74
8.2 Ausblick 75
8.2.1 Speicherung von Prozessmodellen 75
8.2.2 Overlay Erklärungen 76
8.2.3 Dynamische Einstellungen 76
8.2.4 Ausblenden von Swimlanes 76

Literatur **77**

Listings

2.1 React Klassen Komponente 23
2.2 React funktionale Komponente mit Hooks 24

6.1 Funktion zum Hochladen der Datei 54
6.2 Initialisierung von Bpmn.io und Selektion der BPMN 2.0 Elemente . . 56
6.3 Selektion spezifischer BPMN 2.0 Elemente 57
6.4 Rendern der Pools 59
6.5 Markierung der Swimlanes 60
6.6 Positionierung der Overlays 61
6.7 Hinzufügen und Entfernen der Overlays 62
6.8 Hinzufügen von Elementen und Verbindungen zum Prozessmodell . 64
6.9 Funktion zur Einstellung der Detailstufe über den Regler 65
6.10 Funktionen zur Einstellung der Sichtbarkeit von Annotationen 66

Abbildungsverzeichnis

2.1 Aktivität 6
2.2 Pool und Lanes 7
2.3 Sequenzfluss 7
2.4 Nachrichtenfluss 8
2.5 Startereignis 8
2.6 Endereignisse 9
2.7 Zwischenereignis 9
2.8 Prozessmodell mit XOR-Gateway 10
2.9 Prozessmodell mit AND-Gateway 11
2.10 Prozessmodell mit OR-Gateway 11
2.11 Ausschnitt eines Prozessmodell mit Datenobjekt und Darstellung eines Datenspeichers 12
2.12 Verwendung einer Annotation 13
2.13 Beispiel eines komplexen Prozessmodells 14
2.14 Bpmn-js Architektur: Komponentensicht. Abbildung basiert auf [5]. . 18

5.1 Client-Server Architektur. Abbildung basiert auf [51]. 39
5.2 Komponentenübersicht der Webanwendung 40
5.3 Sequenzdiagramm React Single-Page Applikation 42
5.4 Mockup der Startseite 43
5.5 Ablauf 'Datei hochladen' als BPMN 2.0 Prozessmodell 44
5.6 Mockup des Hauptbildschirms 46
5.7 Konzept einer Erklärung in Form von Benennung 49
5.8 Konzept der farblichen Markierung von Swimlanes 50
5.9 Ablauf der Anwendung als BPMN 2.0 Prozessmodell aus Systemperspektive 51

6.1 Komponente für das Hochladen von BPMN 2.0 XML-Dateien, inklusive eingeblendeter Fehlermeldung 53
6.2 Sicht auf den Startbildschirm der Anwendung 55
6.3 Komponente für die Auflistung und farbliche Markierung von Swimlanes, inklusive Beispiel Prozessmodell 58
6.4 Darstellung der Overlays im Prozessmodell 60
6.5 Feinsteuerung der Sichtbarkeit einzelner Elemente 63
6.6 Regler zur Steuerung der Detailstufe 64
6.7 Betrachtung eines Prozesses auf niedrigster Detailstufe 67
6.8 Prozessmodell für das Erstellen und Versenden einer Rechnung . . . 68
6.9 Betrachtung eines Prozesses auf Detailstufe 0 68
6.10 Betrachtung eines Prozesses auf Detailstufe 1 69
6.11 Betrachtung eines Prozesses auf Detailstufe 2 69
6.12 Betrachtung eines Prozesses auf Detailstufe 3 70

Tabellenverzeichnis

4.1 Funktionale Anforderungen mit Priorisierung 31
4.2 Nicht-funktionale Anforderungen mit Priorisierung 35

7.1 Abgleich der funktionalen Anforderungen mit Verweis auf das jeweilige Konzept 71
7.2 Abgleich der nicht-funktionalen Anforderungen mit Verweis auf das jeweilige Konzept 72

1 Einleitung

Dieses Kapitel motiviert zunächst, wieso es notwendig ist, Prozessmodelle dynamisch während der Betrachtung verändern zu können. Daraufhin wird aufgezeigt, was das Problem dabei ist, dass Prozessmodelle einen Vorgang statisch darstellen. Des Weiteren erfolgt die Vorstellung der Zielsetzung dieser Arbeit. Abschließend wird ein Überblick über den Aufbau der Arbeit gegeben.

1.1 Motivation

Heutzutage ist es selbstverständlich geworden, Geschäftsprozesse mit Notationen wie beispielsweise Business Process Model and Notation (BPMN) zu nutzen, um ein besseres Verständnis über Aufgaben und Abläufe zu bekommen [24]. Diese Prozessmodelle beschreiben typischerweise in grafischer Form Aktivitäten, Ereignisse und Kontrollfluss Logik eines Geschäftsprozesses [42]. Außerdem ist es möglich, zusätzliche Informationen, beispielsweise Erklärungen in Form von Annotationen, in diese Prozessmodelle einzufügen. Diese zusätzlichen Informationen sind für Anfänger/-innen hilfreicher als für Experten und Expertinnen [43]. Wie eine Studie [22] zeigt, sorgen Prozessmodelle für ein besseres Verständnis von Geschäftsprozessen und dienen generell zur Verbesserung dieser. Das Verständnis von Prozessmodellen ist laut Mendling et al. [33] abhängig von persönlichen Faktoren, wie beispielsweise theoretischer Erfahrungen von Prozessmodellen. Das bedeutet, dass es erstrebenswert ist, Personen mit weniger Erfahrungen bei der Betrachtung zu unterstützen. Es sollte also das Ziel sein, Prozessmodelle so verständlich wie möglich für Betrachtende zu gestalten, dass möglichst viele Personen und damit auch das Unternehmen davon profitieren können.

1.2 Problemstellung

Ein großer Nachteil bei der Modellierung von Prozessmodellen ist, dass die resultierenden Prozessmodelle statisch sind [21]. Das bedeutet, dass diese ausschließlich das abbilden, was der Modellierer auch tatsächlich modelliert hat [2]. Es ist weder möglich, Informationen aus dem Prozessmodell auszublenden, noch ist es möglich, zusätzliche Informationen anzuzeigen. Außerdem muss dieses statische Prozessmodell für sämtliche Betrachtenden verständlich sein [47]. Dabei spielt es keine Rolle, welche Erfahrungen diese mitbringen und wie vertraut sie mit BPMN 2.0 sind. Benötigt eine Person für das Verständnis mehr Informationen, die eventuell per Annotationen hinzugefügt werden könnten, sorgt diese zusätzliche Annotation hingegen dafür, dass eine andere Person abgelenkt wird und die Konzentration verliert. Außerdem kann die Informationsmenge für Betrachtende überfordernd sein ohne visuelle Führung [44], wie beispielsweise einer farblichen Hervorhebung von Organisationen.

Ein weiterer Punkt ist die Darstellung von Datenflüssen mit Datenobjekten und Datenspeicher. Nicht alle Betrachtenden benötigen zwangsläufig eine Sicht auf den Datenfluss. Es ist durchaus denkbar, dass sich manche Betrachtende ausschließlich einen Überblick über die Aufgaben verschaffen wollen.

Tufte schreibt in [52], dass es herausfordernd ist, eine Benutzerfreundliche und leicht zu verstehende Darstellung von Prozessmodellen zu finden. Aktuelle Lösungen von dynamischen Prozessmodellen [26, 25] beschränken sich auf eine Vorverarbeitung. Damit ist es zwar möglich, Prozessmodelle bis zu einem gewissen Grad an die Betrachtenden anzupassen, dennoch bleibt das daraus resultierende Prozessmodell wiederum statisch und beinhaltet auch nicht mehr alle Informationen des ursprünglichen Modells. Aktuelle Lösungen bieten demnach nicht die Möglichkeit, Geschäftsprozesse während der Betrachtung dynamisch zu verändern.

1.3 Zielsetzung

Ziel dieser Arbeit ist die Konzipierung und Realisierung einer Anwendung, mit der Benutzende den Detailgrad des Prozessmodells dynamisch während der Betrachtung ändern können. Das bedeutet, dass der Benutzende die Menge an Informationen selbst regulieren und je nach Bedarf an sich selbst anpassen kann. Dies geschieht, indem die Sichtbarkeit einzelner BPMN 2.0 Elemente wie Annotationen, Datenspeicher, Datenobjekte und Nachrichtenflüsse eingestellt werden kann. Die Sichtbarkeiten müssen per Schieberegler veränderbar sein. Ergänzend dazu muss die Möglichkeit bestehen, zusätzliche Informationen per Overlay hinzuzufügen. Außerdem sollen die in einem Prozessmodell enthaltenen Swimlanes dynamisch aufgelistet werden. Des Weiteren soll es möglich sein, einzelne Pools und Lanes farblich zu markieren und damit hervorzuheben.

1.4 Gliederung der Arbeit

Kapitel 2 erläutert die verwendeten Technologien und Konzepte, die zum Verständnis dieser Arbeit beitragen. Im Fokus von **Kapitel 3** steht die Vorstellung von Arbeiten, die eine ähnliche oder sogar die gleiche Thematik behandeln. Außerdem wird diese Arbeit eingeordnet. **Kapitel 4** stellt die Anforderungen an die Anwendung vor. Dabei sind diese in funktionale und nicht-funktionale Anforderungen unterteilt. **Kapitel 5** erläutert die Konzeption zur Anwendung und stellt die Architektur sowie die Komponenten vor. Während **Kapitel 6** ausgewählte Aspekte der Implementierung und die dazugehörige Benutzeroberflächen aufzeigt. In **Kapitel 7** werden die aus **Kapitel 4** vorgestellten Anforderungen abgeglichen. Abschließend wird in **Kapitel 8** diese Arbeit zusammengefasst und ein Ausblick über potenzielle zukünftige Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden verwendete Technologien und Konzepte erläutert, die zum Verständnis dieser Arbeit beitragen. Zuerst wird BPMN 2.0 vorgestellt und die wichtigsten Basis Modellierungselemente erklärt. Anschließend wird Node.js und die dazugehörigen Komponenten vorgestellt. Daraufhin folgt die Vorstellung von bpmn.io, das eine BPMN 2.0 Bibliothek zur Darstellung und Modellierung von Prozessmodellen ist. In den nächsten Unterkapiteln erfolgt ein Überblick über Sass und Bootstrap, die für das Styling der Anwendung sorgen. Das vorletzte Unterkapitel beschäftigt sich mit React, das zur Erstellung der Benutzeroberfläche verwendet wird. Im letzen Unterkapitel wird der Begriff Usability beschrieben und die zugehörige Norm vorgestellt.

2.1 BPMN 2.0

Um die Implementierung und Realisierung verstehen zu können, ist es notwendig, ein Grundverständnis über Business Process Model and Notation (BPMN 2.0) zu haben. Dieses Kapitel soll diese Basis schaffen.

BPMN 2.0 ist ein von der Object Management Group (OMG) entwickelter Standard. Dessen Ziel ist es, eine einheitliche grafische Notation zur Verfügung zu stellen, mit der Geschäftsprozesse abgebildet werden können. Die abgebildeten Prozesse sollen von allen Geschäftsteilnehmern einfach zu verstehen sein. Dabei soll es keine Rolle spielen, welche Position die Anwendende innehaben. Ein weiteres Ziel ist es, dass XML-basierte Sprachen, die für die Ausführung von Geschäftsprozessen erstellt worden sind, visualisiert werden können [38]. Die Visualisierung erfolgt mit einer geschäftsorientierten Notation [38]. BPMN 2.0 ist der De-facto-Standard für die grafische Abbildung von Geschäftsprozessen [7].

2.1.1 Diagrammtypen

BPMN 2.0 stellt drei Diagrammtypen zur Verfügung. Wobei der Verwendungszweck bei jedem dieser Typen ein anderer ist. Folgend werden diese vorgestellt.

- **Collaboration Diagram**

  Zeigt die Prozessstruktur und den Prozessfluss einzelner Teilnehmer, beispielsweise Aktivitäten sowie den Kontrollfluss und Datenfluss zwischen diesen [38].

- **Conversation Diagram**

  Dokumentiert die Kommunikationsschnittstellen zwischen zwei oder mehreren Teilnehmern in einem organisationsübergreifenden Prozess [38].

- **Choreography Diagram**

  Dokumentiert die Interaktionen, beispielsweise der Nachrichtenaustausch zwischen unterschiedlichen Partnern. Dies findet ebenso in einem organisationsübergreifenden Prozess statt [38].

Diese Arbeit beschränkt sich auf das *Collaboration Diagram*, da es der meist verwendete Diagrammtyp ist [1] und zur Modellierung von Geschäftsprozessen dient. Allweyer schreibt außerdem in [1], dass manche Tools und Bücher sich ausschließlich auf Collaboration Diagramme beschränken. Laut Chinosi und Trombetta verwenden ca. 85% der Benutzer/-innen weder das Conversation noch das Choreography Diagramm [7].

2.1.2 Basis Modellierungselemente

In diesem Unterkapitel werden die BPMN 2.0 Kernelemente laut [38] vorgestellt. Zu den Flussobjekten zählt das Ereignis, die Aktivität und das Gateway. Die verbindenden Objekte bestehen aus Sequenzfluss, Nachrichtenfluss und Assoziation. Swimlanes setzen sich aus Pools und Lanes zusammen. Die letzte Kategorie sind die Artefakte, die sich aus Datenobjekten, Annotationen und Gruppen zusammensetzen. Gruppen werden an dieser Stelle vernachlässigt und nicht vorgestellt, da sie nur als visuelle Hilfestellung dienen um Elemente zusammenzufassen.

Der BPMN 2.0 Standard definiert über 100 Elemente [53]. Die Vorstellung sämtlicher Elemente würde über den Rahmen dieser Arbeit hinausgehen und wäre nicht zielführend. Für detailliertere Beschreibungen sowie zur Einsicht nicht behandelter Elemente kann der Standard der OMG [38] konsultiert werden.

Aktivität

Eine Aktivität beschreibt eine Arbeitseinheit, die innerhalb eines Prozesses ausführbar ist [38]. Diese wird durch ein Rechteck repräsentiert, welches abgerundete Ecken hat. Die folgende Abbildung 2.1 visualisiert eine Aktivität.

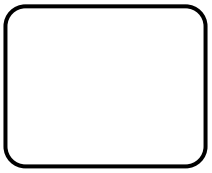

Abbildung 2.1: Aktivität

Die Aktivität kann atomar sowie nicht-atomar sein. Atomar bedeutet, dass die Aktivität nicht weiter in kleinere Arbeitsschritte unterteilbar ist. Eine atomare Aktivität wird als Aufgabe bezeichnet. Benannt wird diese meist mit einer Kombination aus Verb, gefolgt von Substantiv. Beispielsweise könnte in einer Aktivität 'Überprüfe Kontostand' stehen [38]. Nicht-atomar sind sogenannte Subprozesse, die einen Überbegriff darstellen und nochmals in kleinere Arbeitsschritte unterteilbar sind. Diese haben mittig am unteren Rand ein kleines Quadrat mit einem Plus-Symbol [38].

Pool und Lane

Ein Pool repräsentiert eine unabhängige Organisation und modelliert die Ansicht auf den Prozess aus dessen Perspektive. Aktivitäten werden üblicherweise in Pools platziert. Wenn ein Pool keine Elemente enthält wird er als 'Blackbox-Pool' bezeichnet. Dieser lässt ausschließlich Nachrichtenfluss zu. Falls das Diagramm nur eine einzige Organisation abbildet, ist der Pool optional zu verwenden [38].

Lanes sind optionale Unterteilungen einer Organisation in kleinere Einheiten [38]. Beispielsweise ist es möglich einen Pool, der eine Organisation darstellt, mittels Lanes in einzelne Abteilungen zu unterteilen. Lanes sind beliebig tief ineinander schachtelbar [38]. Abbildung 2.2 zeigt einen Pool der zwei Lanes beinhaltet.

Abbildung 2.2: Pool und Lanes

Sequenzfluss

Der Sequenzfluss oder auch oft Kontrollfluss genannt, wird durch einen durchgezogenen Pfeil dargestellt und bestimmt die Ordnung, in der die Aktivitäten ausgeführt werden müssen. Die Aktivitäten werden dann in Pfeilrichtung sequenziell abgearbeitet [38]. Abbildung 2.3 zeigt das Modellierungselement.

Abbildung 2.3: Sequenzfluss

Der Sequenzfluss verbindet Prozessschritte wie Aktivitäten, Gateways und Ereignisse. Er darf die Grenzen eines Pools nicht überschreiten, aber innerhalb eines Pools beliebig über Lane-Grenzen hinweg modelliert werden [38].

Nachrichtenfluss

Ein Nachrichtenfluss wird dafür verwendet, um zwischen den Pools zu kommunizieren, da dort kein Sequenzfluss möglich ist [38]. Dieser wird durch einen gestrichelten Pfeil dargestellt, wie in Abbildung 2.4 zu sehen.

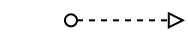

Abbildung 2.4: Nachrichtenfluss

Ein Nachrichtenfluss tritt aus einem sendenden Ereignis oder einer entsprechenden Aktivität aus. Der Nachrichtenfluss kann von einer Aktivität, einem Ereignis oder einem Blackbox-Pool empfangen werden [38].

Startereignis

Ein Startereignis definiert die Stelle, an der ein Prozess oder ein Subprozess beginnt. Ein Ereignis, welches dafür sorgt, dass der Prozess startet, wird Auslöser genannt. Dieser kann beispielsweise eine eingehende Nachricht (Briefsymbol) oder ein bestimmter Zeitpunkt (Uhrsymbol) sein [38]. Dargestellt wird ein Startereignis durch einen einfach gezogenen Kreis, wie in Abbildung 5.4 zu sehen.

Abbildung 2.5: Startereignis

Je nach Auslöser beinhaltet es ein anderes Symbol [38]. Das Startereignis in der Mitte erwartet eine eingehende Nachricht. Während das auf der rechten Seite durch ein zeitliches Ereignis ausgelöst wird [38].

Endereignis

Ein Endereignis definiert die Stelle, an dem ein Teil eines Prozesses oder der komplette Prozess abgeschlossen ist [38]. Es wird durch einen fett gezeichneten Kreis dargestellt, wie in Abbildung 2.6 zu sehen.

Abbildung 2.6: Endereignisse

Je nach Typ ändert sich das Symbol. Das Endereignis in der Mitte mit dem Briefsymbol sendet eine Nachricht sobald es erreicht wurde und hat somit einen ausgehenden Nachrichtenfluss. Das auf der rechten Seite mit dem Punkt terminiert einen Prozess komplett. Ein Endereignis hat keinen ausgehenden Sequenzfluss [38].

Zwischenereignis

Ein Zwischenereignis definiert die Stellen, zwischen Start und Ende, an denen ein Ereignis auftritt [38]. Es wird durch einen doppelten Kreis gekennzeichnet wie in Abbildung 2.7 zu sehen.

Abbildung 2.7: Zwischenereignis

Zwischenereignisse können je nach Typ ein- sowie ausgehende Sequenz- und Nachrichtenflüsse haben [38]. Auf der linken Seite ist ein sendendes Zwischenereignis zu sehen und auf der rechten Seite ein empfangendes zeitbasiertes. Gelangt ein Prozess zu einem empfangenden Ereignis, wartet dieser solange bis das Ereignis eintrifft [38].

Exklusives Gateway

Das exklusive Gateway, auch XOR-Gateway genannt, wird verwendet, um den Prozessverlauf über sich gegenseitig ausschließende Pfade zu verzweigen (XOR-Split) bzw. wieder zu verknüpfen (XOR-Join). Es kann jeweils nur ein Pfad gewählt werden [38].

Abbildung 2.8 zeigt die Verwendung eines XOR-Gateways in einem Prozessmodell. Eine Person hat eine Rechnung erhalten und möchte diese begleichen. Nach

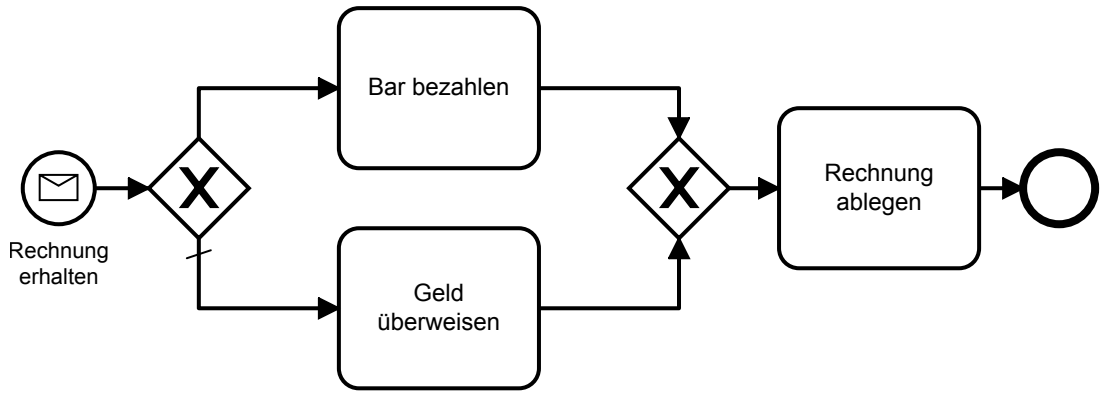

Abbildung 2.8: Prozessmodell mit XOR-Gateway

dem Erhalt der Rechnung startet der Prozess und es folgt die Entscheidung, wie diese beglichen werden soll, entweder Bar oder per Überweisung. Der Pfad zum Geld überweisen hat einen Schrägstrich. Dieser Pfad ist der Standardweg, sofern kein alternativer Pfad infrage kommt. Nachdem die Aktivität ausgeführt ist, sorgt ein XOR-Join dafür, dass die getrennten Pfade wieder vereint werden. Die explizite Verknüpfung per XOR-Gateway ist optional [38], jedoch zu empfehlen. Am Schluss wird die Rechnung abgelegt und der Prozess endet.

Paralleles Gateway

Das parallele Gateway oder auch als AND-Gateway bezeichnet, spaltet den Sequenzfluss in beliebig viele Pfade auf, die dann alle parallel ausgeführt werden. Ein AND-Split spaltet den Prozessfluss, während ein And-Join solange auf die parallel ausgeführten Aktivitäten wartet bis alle abgeschlossen sind [38]. Parallelität bedeutet jedoch nicht, dass die Aktivitäten alle zwingend zeitgleich ausgeführt werden, sondern lediglich, dass die Reihenfolge nicht vorgegeben ist [49].

Abbildung 2.9 zeigt ein Beispiel eines AND-Gateway Prozesses. Nachdem der Prozess gestartet ist, erfolgt eine Aufteilung durch ein AND-Split in zwei Pfade. Die Ware wird versendet, während auf die Bezahlung gewartet wird. Sobald beide Aktivitäten erfolgreich abgeschlossen sind, kann die Bestellung archiviert werden und der Prozess ist abgeschlossen.

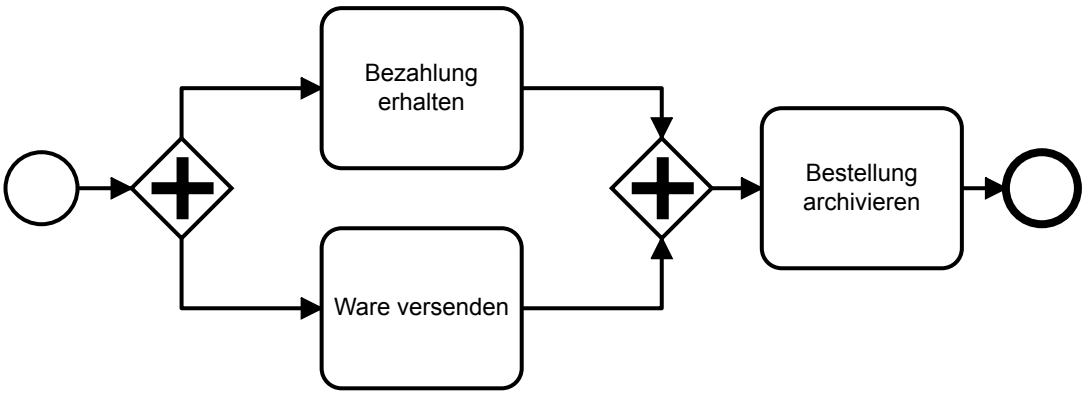

Abbildung 2.9: Prozessmodell mit AND-Gateway

Inklusives Gateway

Das inklusive oder auch OR-Gateway genannt, ähnelt vom Aufbau her dem XOR-Gateway. Ein OR-Split spaltet den Prozessfluss in unterschiedliche Pfade. Allerdings ist es hier möglich, dass nicht nur ein Pfad, sondern mehrere nach erfolgter Entscheidung durchlaufen werden. Der OR-Join verknüpft die Pfade wieder miteinander zu einem Sequenzfluss [38].

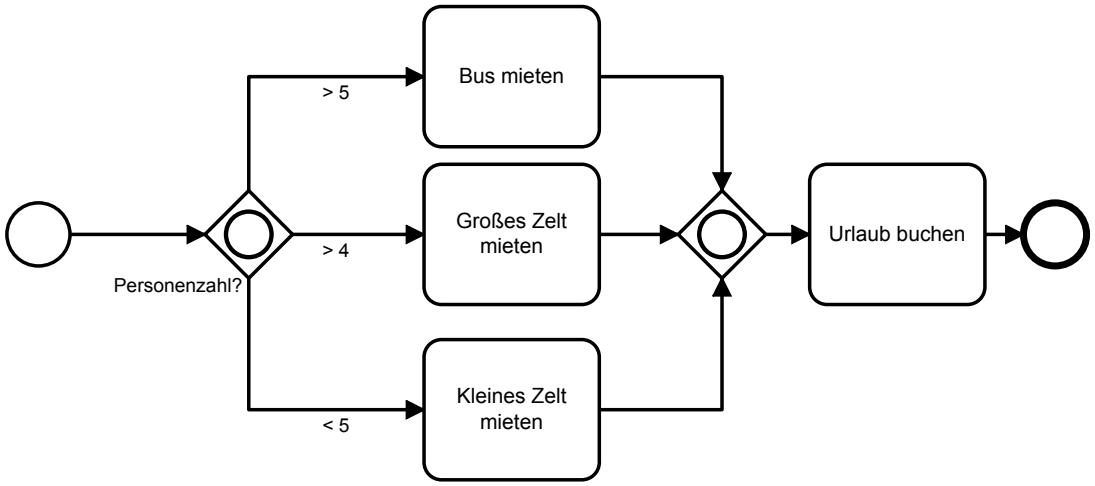

Abbildung 2.10: Prozessmodell mit OR-Gateway

Abbildung 2.10 zeigt ein Beispiel eines OR-Gateway-Prozesses. Nachdem der Prozess der Urlaubsplanung gestartet ist, erfolgt eine Aufteilung in verschiedene Pfade abhängig von der Personenzahl. Wenn weniger als fünf Personen mitkommen, muss nur ein kleines Zelt gemietet werden. Sobald die Anzahl fünf beziehungsweise sechs beträgt, muss ein großes Zelt oder sogar ein Bus gemietet werden. Sobald alle gewählten Pfade durchlaufen sind, wird der Sequenzfluss verknüpft, der Urlaub kann gebucht werden und der Prozess endet.

Assoziation, Datenobjekt und Datenspeicher

Datenobjekte repräsentieren Informationen im Kontext eines Prozesses. BPMN 2.0 macht keine Einschränkungen, was diese enthalten dürfen. Sie repräsentieren einzelne Werte wie Strings oder Booleans, komplexe physikalische Entitäten wie Briefe oder Rechnungen oder digitale Entitäten wie E-Mails oder Dateien. Aktivitäten können Datenobjekte mittels gerichteten oder ungerichteten Assoziationen schreiben oder lesen. Die Pfeilrichtung gibt vor, ob ein Datenobjekt als Eingabe oder Ausgabe dient. Ungerichtet bedeutet sowohl Ein- als auch Ausgabe. Abbildung 2.11 zeigt auf der linken Seite ein Datenzugriff. Die Aktivität *Rechnung erstellen* erzeugt ein Datenobjekt (Rechnung) und die darauf folgende Aktivität liest dieses ein und überprüft die Rechnung.

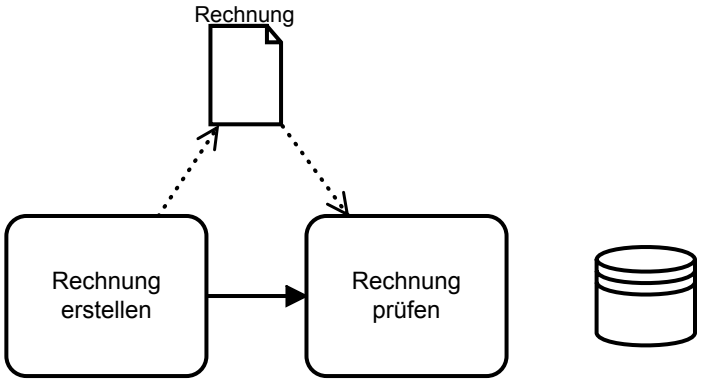

Abbildung 2.11: Ausschnitt eines Prozessmodell mit Datenobjekt und Darstellung eines Datenspeichers

Auf Datenspeicher wird wie bei Datenobjekten mittels Assoziationen zugegriffen. Allerdings repräsentieren sie eine persistente Art, Daten zu speichern, die das Prozessende überdauern [38]. Auf der rechten Seite der Abbildung 2.11 ist ein Datenspeicher zu sehen. Ein Datenspeicher repräsentiert beispielsweise eine Datenbank oder ein anderes Softwaresystem. Datenspeicher sind von außerhalb des Prozesses veränderbar [38]. Während Datenobjekte nur innerhalb des Prozesses veränderbar sind [38].

Das Ein- und Auslesen von Datenobjekten und Datenspeicher mittels Assoziationen wird gemeinhin als Datenfluss bezeichnet [53]. Der Datenfluss beeinflusst nicht den Sequenzfluss und ist somit optional und eher als zusätzliche Information zu sehen. Dieser wird laut Chinosi und Trombetta [7] in Prozessmodellen auch selten verwendet. Obwohl eine Datenflussmodellierung im gewissen Rahmen möglich ist, ist BPMN 2.0 keine Datenfluss-Sprache [38].

Annotation

Annotationen werden verwendet, um zusätzliche Informationen zu vermitteln, die meist nicht mit anderen BPMN Elementen abbildbar sind. Sie können mit jedem beliebigen BPMN Element verbunden werden und sind komplett optional. Dargestellt werden diese durch eine gestrichelte Linie mit einer öffnenden eckigen Klammer wie in Abbildung 2.12 zu sehen [38].

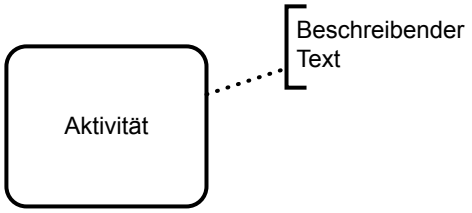

Abbildung 2.12: Verwendung einer Annotation

Innerhalb dieser öffnenden Klammer steht ein beliebiger Text [38]. Gerade für Personen, die einen Prozess zum ersten Mal sehen, können diese zusätzlichen Informationen sehr hilfreich sein und zum Verständnis beitragen.

2.1.3 Prozessmodell Beispiel

Um den Nutzen dieser grafischen Notation zu verdeutlichen und um aufzuzeigen, dass solche Modelle umfangreich und komplex sein können, wird in Abbildung 2.13 ein Prozess vorgestellt zur Verabreichung von Anästhetikum. Nachfolgend ist der Ablauf beschrieben.

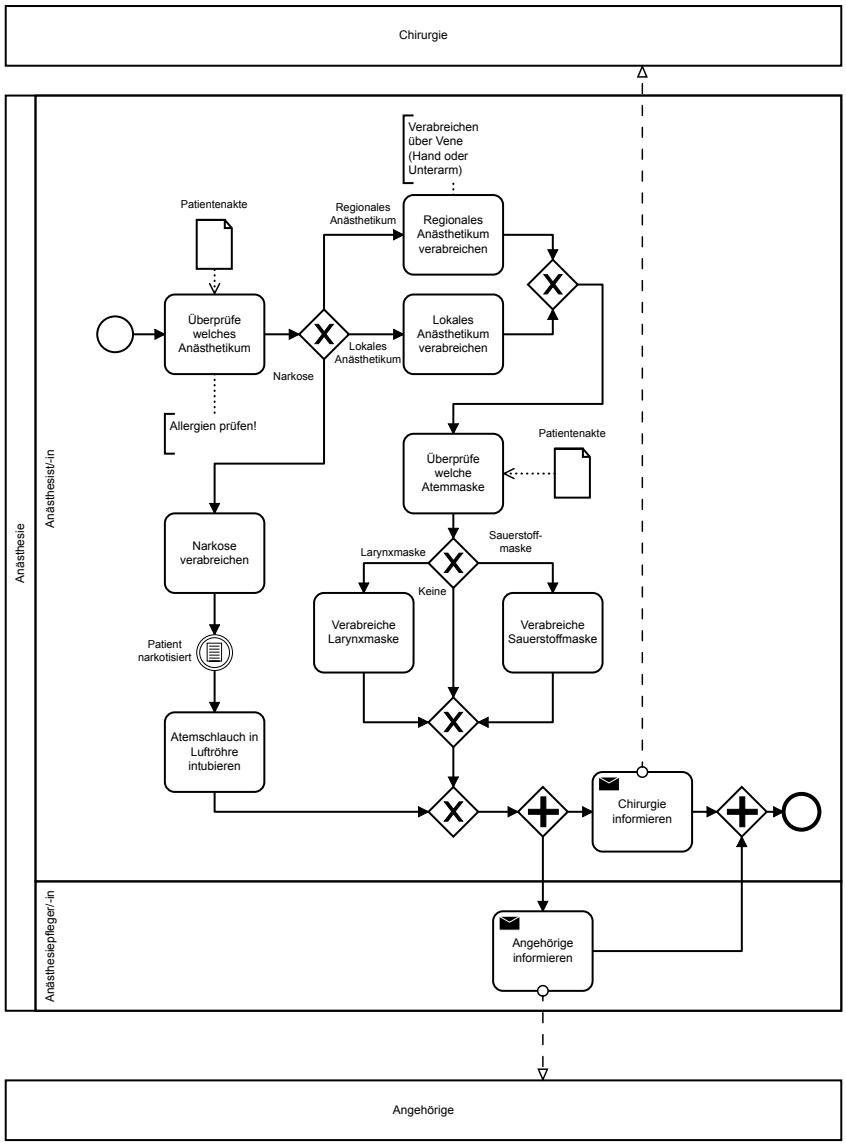

Abbildung 2.13: Beispiel eines komplexen Prozessmodells

Zuerst wird anhand der Patientenakte überprüft, welches Anästhetikum zu verabreichen ist. Wichtige Informationen sind mittels Annotationen an den jeweiligen Aktivitäten angebracht. Bei der Narkose muss der Anästhesist darauf warten, bis der Patient narkotisiert ist. Im Anschluss daran wird dieser intubiert. Bei den anderen Anästhetika erfolgt bei Bedarf eine Gabe von Sauerstoff. Sobald diese Schritte abgeschlossen sind, informiert der*die Anästhesist*in die Chirurgie und der*die Anästhesiepfleger*in die Angehörigen. Damit ist dann der Prozess abgeschlossen. Das abgebildete Modell ist lediglich ein Beispiel. Prozessmodelle können beliebig umfangreich und komplex sein.

2.2 Node.js

Die in dieser Arbeit entwickelte Webanwendung ist mittels Node.js Server, der als Webserver fungiert, gehostet und wird deshalb im Folgenden vorgestellt. Node.js ist eine JavaScript Laufzeitumgebung die dafür gebaut ist, Full-Stack Anwendungen zu erstellen [40]. Diese ist Open-Source und plattformunabhängig [40]. Mittels Node.js ist es möglich, JavaScript Code nicht nur clientseitig, sondern auch serverseitig zu benutzen [20].

2.2.1 Npm

Der Node package manager (npm) ist der De-facto-Standard Paketmanager von Node.js [20]. Mit diesem haben Entwickler/-innen zugriff auf von anderen entwickelten Modulen aus dem *npm package repository* [20]. Diese einzelnen Module bilden die Bausteine, die verwendet werden, um eine Anwendung zu erstellen [20]. Solche Module sind mit dem Befehl *npm install Paketname* zur Anwendung hinzufügbar [40] und tauchen dann daraufhin in einer Datei namens *package.json* auf [40]. Diese Datei beschreibt das Projekt und dessen Abhängigkeiten [40]. Um die ausgewählten Pakete zu installieren, muss der Befehl *npm install*, in dem Ordner, in der die Datei *package.json* liegt, ausgeführt werden [40]. Dabei landen die Dateien in einem Ordner namens *node_modules* [20]. Um hinzugefügte Module in der Anwendung zu verwenden, muss sich in den dafür vorgesehen Dateien ein *import* Befehl befinden [20]. Innerhalb der *package.json* Datei ist es üblich, auf Skripte zu verweisen, die dazu da sind, um beispielsweise die Anwendung zu starten [20]. Wenn eine solches Skript eingebunden ist, wird die Anwendung über den Befehl *npm start* ausgeführt [20].

2.2.2 Npx

Npx ist ein Tool, um Node.js Pakete auszuführen [20]. Es ist seit npm Version 5.2 fester Bestandteil der mitgelieferten CLI [20]. Genutzt wird npx um globale Module auszuführen, ohne diese dauerhaft speichern zu müssen [17]. Das ist insbesondere sinnvoll bei Modulen, die selten oder sogar nur einmal gebraucht werden, da

diese beim nächsten verwenden ohnehin auf die neueste Version gebracht werden müssen [17]. Ein weiterer Verwendungszweck ist das Ausführen von Modulen, die ausschließlich kurz getestet werden sollen und dafür nicht dauerhaft installiert sein müssen [17].

2.3 Bpmn.io

Um ein BPMN 2.0 Prozesmodell im Browser darzustellen, wird ein Framework benötigt, welches XML-Dateien zu grafischen Abbildungen umwandelt. Das Framework, welches für die Realisierung verwendet wird, ist bpmn.io. Dieses ist eine von Camunda entwickelte Open-Source Bibliothek um BPMN 2.0 Prozessmodelle zu modellieren und zu betrachten [5]. Camunda stellt mit bpmn.io eine Bibliothek zur Verfügung die individualisierbar und erweiterbar ist [5].

2.3.1 Hauptkomponenten

Bpmn.io besteht aus drei großen Hauptkomponenten [4], die folgend kurz beschrieben sind.

- **Bpmn-js**

  Wird verwendet um BPMN 2.0 Diagramme im Browser darzustellen und zu modellieren [4].

- **Dmn-js**

  Wird verwendet, um Decision Model and Notation (DMN) Entscheidungsdiagramme und Entscheidungstabellen im Browser anzuzeigen und zu modellieren [4].

- **Cmmn-js**

  Wird verwendet, um Case Management Model and Notation (CMMN) Diagramme im Browser anzuzeigen und zu modellieren [4].

In dieser Arbeit liegt der Fokus auf BPMN 2.0 und somit in der Verwendung von bpmn-js, daher wird folgend dieses Toolkit näher betrachtet.

2.3.2 Bpmn-js

Bpmn-js ist ein webbasiertes BPMN 2.0 Modellierungs- und Betrachtungswerkzeug [5]. Es ist in JavaScript geschrieben und benötigt kein Server Backend [5]. Dadurch ist es einfach in Webanwendungen einzubinden und zu verwenden [5]. Das Toolkit bietet zum einen die Möglichkeit, Prozessmodelle anzuzeigen und mit zusätzlichen Daten anzureichern, zum anderen Prozessmodelle zu editieren und zu erstellen [5].

Bpmn-js kann auf zwei verschieden Arten in die Webanwendung eingebunden werden. Eine Möglichkeit ist das Einbinden eines gebündelten Paketes, das sich entweder lokal auf der Festplatte befindet oder per Content Delivery Network (CDN) beziehbar ist [5]. Die andere Möglichkeit, die unter Verwendung eines Node-Servers am sinnvollsten ist, ist die Einbindung als Modul über npm [5]. Diese Version ermöglicht es auch, die Bibliothek zu manipulieren und an die eigenen Bedürfnisse anzupassen [5].

Bpmn-js besteht im wesentlichen aus drei Hauptkomponenten, die in Abbildung 2.14 zu sehen sind.

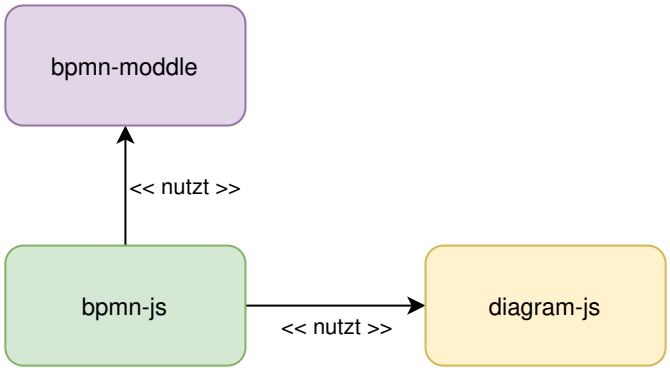

Abbildung 2.14: Bpmn-js Architektur: Komponentensicht. Abbildung basiert auf [5].

Bpmn-js verwendet diagram-js und bpmn-moddle, die jeweils eigenständige Bibliotheken sind [5]. Bpmn-moddle nutzt das BPMN 2.0 Metamodell und bietet die Möglichkeit, BPMN 2.0 XML-Dateien einzulesen und zu schreiben [5]. Während des Imports wird die XML-Datei in ein Javascript Objektbaum geparst und anhand des Metamodells validiert [5]. Der Vorgang wird während des Speichervorgangs rückwärts durchlaufen und am Ende resultiert das Prozessmodell als XML-Datei [5]. Diagram-js ist dafür zuständig, Diagramme anzuzeigen und zu manipulieren [5].

Es erzeugt dabei ein Datenmodell, das aus einfachen Verbindungen und Formen besteht [5]. Außerdem bietet es die Möglichkeit, mit diesen Elementen zu interagieren und stellt zusätzliche Tools wie Overlays zur Verfügung [5]. Bpmn-js selbst definiert die BPMN 2.0 Eigenschaften und baut auf den beiden vorher genannten Komponenten auf [5]. Die Eigenschaften sind hierbei das Erscheinungsbild, Modellierungsregeln und das zur Verfügung stellen der Modellierungswerkzeuge [5].

Bpmn-js ist auf drei verschiedene Arten verwendbar, die all einen unterschiedlichen Funktionsumfang besitzen. Die folgende Auflistung gibt darüber einen Überblick.

- **Viewer**

  Wird verwendet, um BPMN 2.0 Diagramme darzustellen [5]. Bietet den geringsten Funktionsumfang.

- **NavigatedViewer**

  Wird verwendet, um BPMN 2.0 Diagramme darzustellen und um zu navigieren. Das bedeutet, dass das Prozessmodell zusätzlich zur Ansicht auch verschiebbar und zoombar ist [5]. Das ist insbesondere bei umfangreichen und großen Prozessmodellen von Vorteil.

- **Modeler**

  Wird verwendet, um BPMN 2.0 Diagramme zu bearbeiten und zu erstellen [5]. Der Modeler bietet dazu Modellierungswerkzeuge an.

In dieser Arbeit liegt der Fokus auf der Anzeige von BPMN 2.0 Prozessmodellen und nicht auf der Modellierung. Des Weiteren muss die Anwendung auch in der Lage sein, umfangreiche und große Prozessmodelle bedienungsfreundlich darzustellen.

2.4 Sass

Syntactically Awesome Stylesheets (Sass) ist eine Stylesheet-Sprache, die als Erweiterung von Cascading Style Sheets (CSS) fungiert und zusätzliche Funktionen mit sich bringt, die mit regulärem CSS nicht möglich sind [15]. Sass ist ein CSS-Präprozesssor, der in Sass geschriebenen Code in CSS kompiliert [30]. Das bedeutet, dass ein Stylesheet, welches mit Sass geschrieben ist, vor der Verwendung in CSS umgewandelt werden muss. Es ist zudem abwärtskompatibel mit sämtlichen CSS Versionen [15]. Somit ist auch die Verwendung von klassischem CSS Code möglich [30]. Des Weiteren unterstützt Sass zwei unterschiedliche Syntaxen, die folgend näher betrachtet werden.

- **.sass:** Die originale Syntax, orientiert sich an der Haml Skriptsprache [30]. Diese verwendet Einrückungen für Codeblöcke und Zeilenumbrüche, um Regeln voneinander zu separieren [30]. Diese Syntax verwendet *.sass* als Dateierweiterung [30].
- **.scss:** Die neuere Syntax verwendet eine Blockformatierung, die der von CSS stark ähnelt [30]. Geschweifte Klammern grenzen Codeblöcke ab und Semikolons separieren einzelne Codezeilen innerhalb eines Blocks [30]. Für diese Syntax wird die Dateierweiterung *.scss* verwendet [30].

Sass bietet die Möglichkeit, Variablen zu definieren. Diese beginnen mit einem Dollarzeichen, gefolgt von der Bezeichnung [15]. Die Variable kann anschließend beliebig oft eingesetzt werden [15]. Das ist hilfreich, wenn beispielsweise ein systemweites Farbschema existiert und damit sichergestellt werden kann, dass alle verwendeten Elemente denselben Farbton haben. Außerdem ermöglichen Schleifen, Bedingungen und Funktionen Logik innerhalb eines Stylesheets zu verwenden [15]. Beispielsweise ist es dadurch möglich, das Styling von Elementen anhand von Bedingungen per Funktionen anzupassen. Die Verschachtelung von Selektoren ist ein weiteres Merkmal, wieso Sass so beliebt ist [15]. Damit ist die Möglichkeit gemeint, Regeln innerhalb anderer Regeln zu schreiben um Selektoren zu verkürzen und Wiederholungen zu vermeiden[15]. Eine Schachtelung von beliebig vielen Elementen mit einer beliebigen Tiefe ist erlaubt [15].

Diese zusätzlichen Funktionen von Sass sorgen dafür, dass die Stylesheets weniger Redundanzen enthalten und eine bessere Struktur [15]. Dadurch sind diese

vereinfachter, lesbarer und besser wartbar [15]. Nicht vorgestellte Funktionalitäten, die in dieser Arbeit auftreten, sind in der Realisierung bei Bedarf näher erläutert.

2.5 Bootstrap

Bootstrap ist eines der bekanntesten Open-Source Frontend Frameworks für Webanwendungen [41]. Es besteht aus einer Sammlung von CSS und JavaScript Code Dateien, mit der die Webanwendung responsive konstruierbar ist [31]. Das bedeutet, dass es für unterschiedliche Bildschirmgrößen ausgelegt ist. Durch die Verwendung dieser vorgefertigten Codeausschnitte müssen die Entwickelnden deutlich weniger CSS Code schreiben. Bootstrap kann manuell heruntergeladen, über ein CDN zur Laufzeit geladen oder über npm installiert werden [28].

Bootstrap hat mittels CSS und JavaScript Komponenten definiert, die häufig zum Einsatz kommen [15]. Diese haben ein vordefiniertes Aussehen sowie Verhalten und sind für bestimmte Anwendungszwecke konzipiert. Ein Beispiel für solch eine Komponente ist ein Fortschrittsbalken. Die Komponenten sind über Klassenselektoren aufrufbar [15]. Wenn eine Komponente in der Anwendung verwendet werden soll, wird dem betreffenden HTML Element innerhalb des Klassenattributes der entsprechende Klassennamen der Komponente zugewiesen [15].

Bootstrap bietet Möglichkeiten, das vordefinierte Standard Aussehen zu verändern [15]. Alle CSS-Regeln und JavaScript Funktionalitäten sind überschreibbar [41]. Außerdem bleibt die Möglichkeit bestehen, eigene Regeln zu definieren und bei Bedarf das Styling manuell zu übernehmen [41]. Beispielsweise ist es möglich, das Aussehen des vorher erwähnten Fortschrittsbalkens anzupassen oder sogar komplett zu übernehmen.

2.6 React

React ist ein anpassungsfähiges und vielseitiges JavaScript Framework, das verwendet wird, um Benutzeroberflächen zu gestalten [16]. Entwickelt wurde es 2011 von Jordan Walke, einem Softwareentwickler bei Facebook [40]. Entwickler/-innen nutzen es, um große JavaScript lastige Single-Page Anwendungen zu erschaffen oder eben überraschend kleine Plug-ins [16]. Single-Page Anwendungen sollen sich von der Bedienung so anfühlen und so aussehen, als wären es native Anwendungen [9]. Dafür wird initial eine HTML-Seite geladen, die dann dynamisch zur Laufzeit stetig angepasst wird, ohne dass weitere synchrone Anfragen an den Server gestellt werden müssen [9]. Diese HTML-Seite besitzt dazu ein Wurzelelement, das als Container für die Anwendung fungiert [9].

React ist komponentenbasiert und ermöglicht das Einbinden der Komponenten mittels JSX [40]. JSX ist eine tag-basierte JavaScript Syntaxerweiterung, die vom Aussehen HTML stark ähnelt [40]. React benötigt intern Webpack und Babel [40]. Hierbei bündelt Webpack alle JavaScript Dateien zu einem Bündel [40]. Babel hingegen transpiliert React Code, der von Browsern nicht unterstützt wird, beispielsweise JSX, zu purem JavaScript [40].

2.6.1 Components

Eine Benutzerschnittstelle besteht für gewöhnlich aus vielen einzelnen Teilen, wie beispielsweise Navigationsleisten, Tabellen oder Listen [40]. In React werden diese Teile als Components bezeichnet [40]. Diese erlauben es, die selbe Struktur zu verwenden und sie dann mit verschiedenen Datensätzen zu befüllen, ohne an der Komponente selbst etwas zu ändern [40]. Listing 2.1 zeigt eine Klassenkomponente, die eine Überschrift erzeugt. In Zeile 4 wird die Klassenkomponente *Ueberschrift* erstellt. Im Konstruktor in Zeile 5 werden die *props* übergeben und anschließend als Zustand gespeichert. In der *render* Funktion in Zeile 10 wird der Ausdruck in der geschweiften Klammer durch den im Konstruktor zugewiesenen Namen ersetzt und anschließend die Überschrift zurückgegeben. In der *render* Funktion von *ReactDOM* in Zeile 15, wird die Komponente *Ueberschrift* mit dem Namen 'Max', welcher über das Attribut als prop in die Komponente gelangt, in den HTML Container mit der Id *'root'* gerendert.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3
4  class Ueberschrift extends React.Component {
5    constructor(props) {
6      super(props);
7      this.state = {name: props.name};
8    }
9
10   render() {
11     return <h1>Hallo, {this.state.name}!</h1>;
12   }
13 }
14
15 ReactDOM.render(
16     <Ueberschrift name="Max"/>,
17     document.getElementById("root")
18 );
```

Listing 2.1: React Klassen Komponente

Der Vorteil dieser Komponenten ist, dass diese beliebig oft verwendbar sind. Es ist möglich, die props dynamisch per Attribut zu steuern und mit der gleichen Komponente jedes Mal ein unterschiedliches Ergebnis zu erhalten, ohne diese manuell anzupassen. Sobald sich die *props* ändern, wird ein 'neu rendern' ausgelöst [9]. Bei dieser Aktion werden lediglich die Komponenten erneuert, die aktualisiert werden müssen. Dadurch fällt das Auswechseln der Daten im Code weg [9].

Komponenten haben ihren eigenen Lebenszyklus und bieten in den einzelnen Phasen verschiedene Methoden an, wovon die meisten nicht genutzt werden und ausschließlich Randfälle abdecken [14]. Um diese Klassenkomponenten weniger umfangreich zu gestalten und in kleinere Komponenten unterteilen zu können, hat Facebook ein Feature mit dem Namen Hooks entwickelt [40], welches im folgenden Unterkapitel vorgestellt wird.

2.6.2 Hooks

Mit React Version 16.8 hat Facebook Hooks vorgestellt [13]. Hooks sind Funktionen, die es erlauben, in den React-Lebenszyklus und in die Zustandshaltung einzugreifen [13]. Seit der Einführung gibt es Komponenten, die Functional Components heißen, diese verwenden die Hooks und sind nicht an Klassen gebunden [13].

Listing 2.2 stellt eine funktionale Komponente vor unter Verwendung von Hooks. Von der Funktionsweise gleicht dieser Ausschnitt dem Ausschnitt aus Listing 2.1. In der ersten Zeile wird der *useState* Hook importiert und steht zur Verwendung bereit. In Zeile 3 wird anstatt einer Klasse eine funktionale Komponente definiert, wobei die *props* einzeln mit ihren Bezeichnern übergeben werden, anstatt als ganzes Objekt. Zeile 4 zeigt die Verwendung des useState um einen lokalen Zustand halten zu können. Dabei wird eine Variable mit dem Namen *name* definiert und eine Funktion, die *setName* heißt. Ausschließlich mit dieser Funktion ist der Zustand von *name* veränderbar. Der Zugriff auf den Namen erfolgt ähnlich zur Klassen Komponente in Zeile 7 über den Bezeichner. Der Ausdruck in Zeile 10 exportiert die Komponente und ermöglicht es auf sie, innerhalb des Projekts zuzugreifen.

```
1  import React, {useState}  from 'react';
2
3  const Ueberschrift = ({ name }) => {
4    const [name, setName] = useState('');
5
6    render() {
7      return <h1>Hallo, {name}!</h1>;
8    }
9  }
10 export default Ueberschrift;
```

Listing 2.2: React funktionale Komponente mit Hooks

Funktionale Komponenten unter Verwendung von Hooks sind schlanker, da sie weniger Code benötigen und moderner sind. Des Weiteren hält eine Klassenkomponente ihren Zustand in einem Objekt, was schnell dazu führt, dass dieser unübersichtlich und schlecht wartbar wird. Aus diesen Gründen und weil es in aktueller Literatur [9, 40, 16] empfohlen wird, sind bei der Realisierung ausschließlich funktionale Komponenten zum Einsatz gekommen.

2.7 Usability

Das Augenmerk bei der Gestaltung der Webanwendung liegt auf der Usability. Deshalb wird in diesem Kapitel erläutert, was Usability überhaupt bedeutet und im Anschluss auf den Standard DIN EN ISO-9241-110 eingegangen, auf den sich die Anforderungen dieser Realisierung stützen. Die Norm 9241 ist ausgewählt, da sie am stärksten etabliert ist [27]. Abschnitt 110 der DIN EN ISO-9241 beschäftigt sich dabei mit dem Thema Software-Ergonomie [27] und steht deshalb im Fokus.

Nielsen beschreibt in [36] Usability als Qualitätsmerkmal, wie schnell erlernbar und wie leicht etwas zu bedienen ist. Außerdem muss damit das gesteckte Ziel zu erreichen sein. Des Weiteren erwähnt er, dass auch eine möglichst geringe Fehleranfälligkeit des verwendeten Gegenstandes wichtig ist, da auf diesen sonst verzichtet werden kann. Eine modernere und mehr auf Software bezogene Beschreibung von Usability kommt von Padolsey in [39]. Er beschreibt Usability als etwas, was alle Benutzer/-innen betrifft, egal welche Rolle sie haben. Dabei sollen alle Funktionen und Interaktionen so hilfreich und einfach zu benutzen wie möglich sein. Es geht dabei nicht nur darum, die Anforderungen der Anwendenden zu erfüllen, sondern eine Nutzererfahrung zu erschaffen, die es ihnen möglich macht, ihre Ziele mit minimaler Anstrengung und Zeitaufwand zu erreichen [39]. Beide Beschreibungen des Begriffes Usability ähneln sich stark, obwohl einige Jahre zwischen beiden liegen und bei Padolsey der Fokus sich auf interaktiven Systemen befindet.

DIN EN ISO-9241-110

Abschnitt 110 mit dem Titel „Grundsätze der Dialoggestaltung“ ist ein Teil der Norm DIN EN ISO-9241 [46]. Dieser Teil behandelt die Gestaltung von interaktiven Systemen unter Berücksichtigung des Ergonomie Aspekts [45]. Dafür sind sieben Grundsätze der Dialoggestaltung aufgestellt und beschrieben, die jedoch unabhängig von bestimmten Dialogtechniken sind [45]. Diese Grundsätze betreffen die Entwicklung von Benutzerschnittstellen jeglicher Art [45]. Sie werden folgend aufgelistet und erläutert.

1. **Aufgabenangemessenheit**

   Ein Dialog ist aufgabenangemessen, wenn dieser die Anwendenen unterstützt, ihre Arbeitsaufgaben effektiv und effizient zu erledigen [37]. Das bedeutet, dass alle Funktionen, die zur Aufgabenbewältigung dienen, vorhanden, einheitlich gestaltet und ausreichend erklärt sind. Des Weiteren sind keine überflüssigen Arbeitsschritte erforderlich und alle benötigten Informationen und Komponenten sind vorhanden und leicht erreichbar.

2. **Selbstbeschreibungsfähigkeit**

   Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich oder den Benutzenden auf Anfrage erklärt wird [37]. Das bedeutet, dass klar ersichtlich ist wo sich die Anwendenden innerhalb der Anwendung befinden und welche weiteren Schritte möglich sind. Außerdem wird diesen immer mitgeteilt, ob durchgeführte Aktionen erfolgreich sind. Komplexe Aufgaben sind ausreichend gut beschrieben.

3. **Erwartungskonformität**

   Ein Dialog ist erwartungskonform, wenn dieser konsistent ist und den Merkmalen der Benutzenden entspricht [37]. Beispielsweise sind Kenntnisstand und Erfahrungen berücksichtigt [37]. Das bedeutet, dass Navigationsmerkmale und Steuerelemente einheitlich und konsistent zu gestalten sind. Da Anwendende unterschiedliche Stufen von Erfahrungen mitbringen muss dies bei der Konzeption berücksichtigt werden.

4. **Lernförderlichkeit**

   Ein Dialog ist lernförderlich, wenn dieser den Benutzenden beim Erlernen des Dialogsystems unterstützt und anleitet [37]. Das bedeutet, dass die Anwendenden durch Erklärungen und Hilfestellungen, die sich im gewissen Rahmen bewegen, unterstützt werden den Dialog zu bedienen. Auch wenn dieser über einen längeren Zeitraum nicht verwendet wird, muss der Wiedereinstieg leicht fallen. Neue und bislang unbekannte Funktionen sollen ohne negative Konsequenzen durch ausprobieren erlernbar sein.

5. **Steuerbarkeit**

   Ein Dialog ist steuerbar, wenn die Benutzenden in der Lage sind, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist [37]. Das bedeutet, dass die Anwendenden so viel Freiheit wie möglich haben sollen, sich innerhalb des Dialoges zu bewegen und diese für jede Aktion beliebig Zeit haben, um diese auszuführen. Getätigte Eingaben können wieder rückgängig gemacht werden.

6. **Fehlertoleranz**

   Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand durch Benutzende erreicht werden kann [37]. Das bedeutet, dass die Anwendenden so weit wie möglich davor geschützt werden, Fehler zu begehen bzw. fehlerhafte Eingaben zu tätigen. Das wäre beispielsweise durch Sicherheitsabfragen oder Eingabevalidierung möglich. Falls die Fehler nicht verhinderbar sind, muss der Dialog dafür sorgen, dass die Anwendenden nur einen geringen Korrekturaufwand haben. Beispielsweise durch das Zurücknehmen per Tastenkombination einer falschen Eingabe.

7. **Individualisierbarkeit**

   Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe, individuelle Vorlieben der Benutzenden und deren Fähigkeiten zulässt [37]. Das bedeutet, dass die Anwendenden die Elemente und Menüpunkte an ihre Bedürfnisse und an ihren Kenntnisstand anpassen und die Menge an Informationen, die auf dem Bildschirm dargestellt sind, einstellen können.

3 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten betrachtet. Es wird auf deren Lösungswege eingegangen und eine Verbindung zu dieser Arbeit gezogen. Unterschiede sind in den jeweiligen Abschnitten am Ende angebracht.

Dani et al. [50] arbeiten die Fachliteratur zum Thema Visualisierung von Geschäftsprozessmodellen in ihrer Arbeit auf. Dabei betrachten sie diese unter zwei Gesichtspunkten. Zum einen beobachten Dani et al. die Ähnlichkeiten der Arbeiten und ordnen die Ausarbeitungen dann in Kategorien ein, wie Visualisierung von Informationen oder Perspektiven. Zum anderen analysieren sie die wissenschaftlichen Veröffentlichungen anhand eines Analyse-Frameworks zum Thema Visualisierung. Diese Arbeit beschäftigt sich mit dem Thema der Visualisierung von Prozessmodellen und vergleicht wissenschaftliche Ausarbeitungen, stellt allerdings keine eigene Methode zur Visualisierung auf.

Es gibt Tools wie proView, die durch Anwendung von Algorithmen die Ansicht auf das Prozessmodell verändern. Das proView Framework [26, 25] wird verwendet, um die Ansicht großer Prozessmodelle zu personalisieren und entsprechend zu abstrahieren. Es verändert die resultierenden Prozessmodelle, indem die Parameter angepasst werden. Durch interne Anwendung von Algorithmen können dadurch beispielsweise irrelevante Teile des Prozesses versteckt oder aggregiert werden. Daraus ergeben sich personalisierte Sichten. Allerdings ist dies nicht während der Betrachtung möglich und es bedarf einer nicht unerheblichen Vorverarbeitung.

Diese Arbeiten [6, 8, 23, 48], beschäftigen sich mit perspektivischen visualisieren von interorganisatorischen Prozessen. Dabei wird die Sichtbarkeit einzelner Details abhängig vom Betrachtenden verändert. Die Veränderung der Sichtbarkeit wird hierbei während der Modellierung verändert und ist Aufgabe des Modellierenden. Prozessdetails sind dabei nicht nur farblich zur Hervorhebung markiert, sondern werden komplett ausgeblendet.

Um Lanes und Pools hervorzuheben, wird in dieser Arbeit eine farbliche Markierung der jeweiligen Swimlanes genutzt. Farbliche Markierungen von Elementen sind auch Bestandteil der wissenschaftlichen Ausarbeitung von Natschläger [35]. Allerdings werden hier Aktivitäten anhand ihrer Wichtigkeit klassifiziert und dementsprechend eingefärbt. Das erweitert das Prozessmodell um eine Logik, wohingegen die Markierung von Swimlanes, in dieser Arbeit lediglich der besseren Orientierung dienen. Kummer et al. [29] beschäftigen sich mit der Verständlichkeit von Prozessmodellen abhängig von der Einfärbung von Elementen. Während Erol [10] generell das Thema Farben in Prozessmodellen aufarbeitet und eine prototypische Implementierung vorstellt.

Es gibt einige Arbeiten [32, 34, 3], die Annotationen verwenden, um zusätzliche Informationen in einen Prozess zu bringen. Diese werden jedoch permanent angezeigt, was unter Umständen die Übersichtlichkeit und somit die Lesbarkeit des Prozesses vermindert.

Die vorgestellten Arbeiten überschneiden sich von der Funktionsweise lediglich in wenigen Aspekten und fokussieren andere Ziele. Auch nach gewissenhafter und umfangreicher Suche konnte keine verwandte Arbeit gefunden werden, die es möglich macht, Swimlanes farblich zu markieren, eine Erklärung der BPMN-Elemente mittels Overlays anzuzeigen sowie den Detailgrad (Annotationen, Datenfluss, Nachrichtenfluss) eines Prozessmodelles zur Laufzeit während der Betrachtung zu verändern und dadurch eine dynamische Anzeige von zusätzlichen Informationen zu ermöglichen. Durch die dynamische Regulierung sind Betrachtende in der Lage, die Menge an Informationen, die in dem Prozessmodell enthalten sind, selbst nach Bedarf zu regulieren. Das ist mit bisherigen Lösungen während der Betrachtung nur eingeschränkt möglich.

4 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen für die Ausarbeitung analysiert und festgelegt. Die Anforderungen sind unterteilt in funktionale- sowie nicht-funktionale Anforderungen. Jede Anforderung ist mit einem eindeutigen Code versehen und ist damit referenzierbar. Für die Priorisierung der einzelnen Anforderungen wird die MoSCoW Priorisierung verwendet. MoSCoW ist ein Akronym und repräsentiert folgende vier Prioritätsgruppen [19].

- **M**ust have: Diese Anforderungen haben höchste Priorität und sind nicht verhandelbar. Falls diese Anforderungen nicht umgesetzt sind, ist das mit einem Scheitern des Projekts gleichzusetzen [19].
- **S**hould have: Diese Anforderungen haben hohe Priorität und sollten umgesetzte werden, insofern sie keine *must have* Anforderungen beeinträchtigen [19].
- **C**ould have: Diese Anforderungen haben niedrige Priorität und werden umgesetzt, wenn noch genügend Kapazitäten frei sind [19].
- **W**on't have: Diese Anforderungen haben die geringste Priorität und werden in der aktuellen Umsetzung nicht berücksichtigt [19].

4.1 Funktionale Anforderungen

Funktionale Anforderungen definieren alle Funktionalitäten, die die Webanwendung erfüllen muss, um die Anwendenden bei der Erfüllung ihrer Aufgaben zu unterstützen. Folgend sind die funktionalen Anforderungen tabellarisch aufgelistet. Im Anschluss daran werden diese näher erläutert. Tabelle 4.1 ist absteigend nach Priorität sortiert und listet alle Anforderungen inklusive Codierung und Priorisierung auf.

Tabelle 4.1: Funktionale Anforderungen mit Priorisierung

Codierung	Anforderung	Priorisierung
F01	Datei einlesen	Must
F02	Format überprüfen	Must
F03	Visualisierung	Must
F04	Einteilung Detaillierungsgrad	Must
F05	Detail-Regler	Must
F06	Sichtbarkeit	Must
F07	Navigieren	Should
F08	Benennung	Should
F09	Detailstufen	Should
F10	Auflistung	Should
F11	Hervorhebung	Should
F12	Ansicht zurücksetzen	Could

F01 - Datei einlesen

Die BPMN 2.0 XML-Datei muss unkompliziert mit gängigen Methoden eingelesen werden. Mehrere Dateien werden nicht akzeptiert. Während des Einlesens der Datei muss den Nutzenden kenntlich gemacht werden, dass ein Ladevorgang stattfindet. Solange der Vorgang andauert, werden weitere Eingaben verhindert.

F02 - Format Überprüfen

Die eingelesene Datei muss auf das Dateiformat hin überprüft werden. Falls die Datei keine valide BPMN 2.0 XML-Datei ist, muss den Anwendenden eine entsprechende Fehlermeldung ausgegeben werden.

F03 - Visualisierung

Die eingelesene und validierte BPMN 2.0 XML-Datei muss mittels BPMN 2.0 Framework visualisiert werden. Dabei muss initial, das gesamte Prozessmodell zu sehen sein.

F04 - Einteilung Detaillierungsgrad

Das Prozessmodell muss in vier Detailstufen eingeteilt werden.

- **Stufe 1:** Repräsentiert das Prozessmodell ohne Datenflüsse (Datenobjekt und Datenspeicher) und Annotationen.
- **Stufe 2:** Repräsentiert das Prozessmodell ohne Annotationen.
- **Stufe 3:** Repräsentiert das Prozessmodell ohne Veränderungen. Das ist die initiale Stufe nach dem rendern.
- **Stufe 4:** Repräsentiert das Prozessmodell ohne Veränderungen. Zusätzlich ist an jedem BPMN Element, das zum ersten Mal im Prozess auftritt, ein Overlay angebracht mit einer Erklärung (Bezeichnung des Elements).

F05 - Detail-Regler

Die Detailstufen müssen mittels Regler einstellbar sein. Bei Veränderung der Detailstufe muss die Sichtbarkeit der jeweiligen BPMN 2.0 Elemente im Prozessmodell angepasst werden.

F06 - Sichtbarkeit

Die Sichtbarkeit einzelner BPMN 2.0 Elemente (Annotation, Datenobjekt, Datenspeicher, Nachrichtenfluss) muss mit einem Kontrollelement veränderbar sein. Sowohl die sichtbaren als auch die unsichtbaren Elemente müssen hierbei klar erkennbar sein.

F07 - Navigieren

Sobald das Prozessmodell gerendert ist, müssen die Anwendenden die Möglichkeit haben, im Prozessmodell zu navigieren (Zoomen und Verschieben).

F08 - Benennung

BPMN 2.0 Elemente sollen bei ihrem ersten Auftreten im Prozessmodell benannt werden.

F09 - Detailstufen

Die zu Auswahl stehenden Detailstufen sollen mit knapper Erklärung visualisiert werden. Dabei ist klar erkennbar, welche Detailstufe derzeit ausgewählt ist.

F10 - Auflistung Swimlanes

Alle Pools und Lanes, die sich im Prozessmodell befinden, sollen dynamisch und kategorisch aufgelistet werden.

F11 - Hervorhebung

Alles Pools und Lanes sollen dynamisch mit einem Kontrollelement farblich hervorgehoben werden können. Dabei sind markierte sowie nicht markierte Swimlanes klar erkennbar.

F12 - Ansicht zurücksetzen

Die Ansicht auf das Prozessmodell kann zurückgesetzt werden. Dabei wird der Vergrößerungsfaktor angepasst und das Prozessmodell zentriert.

4.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen sind eine Art Qualitätseigenschaft oder Bedingung, in welcher die Funktionalitäten erbracht werden sollen. Diese betreffen meist das System als Ganzes. Folgend sind die nicht-funktionalen Anforderungen tabellarisch aufgelistet und im Anschluss daran näher erläutert. Die Anforderungen sind zusätzlich in Usability und System unterteilt. Tabelle 4.2 ist absteigend nach Priorität sortiert und listet alle Anforderungen inklusive Codierung und Priorisierung auf.

Tabelle 4.2: Nicht-funktionale Anforderungen mit Priorisierung

Codierung	Anforderung	Priorisierung
Usability:		
NFU01	Aufgabenangemessenheit	Must
NFU02	Selbstbeschreibungsfähigkeit	Must
NFU03	Lernförderlichkeit	Must
NFU04	Fehlertoleranz	Must
NFU05	Erwartungskonformität	Should
NFU06	Individualisierbarkeit	Could
NFU07	Steuerbarkeit	Could
System:		
NFS01	Plattformübergreifend	Must
NFS02	Single-Page Applikation	Must
NFS03	Wartbarkeit	Must
NFS04	Skalierbarkeit	Should
NFS05	Offline	Should

Usability:

NFU01 - Aufgabenangemessenheit

Die Benutzenden werden bei der Erledigung ihrer Aufgaben effektiv und effizient unterstützt.

NFU02 - Selbstbeschreibungsfähigkeit

Das System bietet eine angemessene Menge an Informationen, die die Benutzung des Systems intuitiv machen.

NFU03 - Lernförderlichkeit

Den Benutzenden fällt es leicht, die Webanwendung in angemessener Zeit zu erlernen. Dabei soll der Fokus auf ausreichender Hilfestellung durch Hilfstexte liegen.

NFU04 - Fehlertoleranz

Die Benutzenden werden auf fehlerhafte Eingaben hingewiesen und das System bleibt weiterhin benutzbar. Fehler haben keine permanenten Konsequenzen und sind leicht behebbar.

NFU05 - Erwartungskonformität

Die Benutzenden sollen in der Lage sein, das System nach einheitlichen Prinzipien zu bedienen. Des Weiteren soll sich die Anwendung an üblicher Bedienbarkeit von Webanwendungen orientieren.

NFU06 - Individualisierbarkeit

Die Menge an Informationen, die auf dem Bildschirm angezeigt werden, kann individualisiert werden

NFU07 - Steuerbarkeit

Die Benutzenden haben die Möglichkeit, die Geschwindigkeit und Richtung des Systems zu bestimmen.

Systemanforderungen:

NFS01 - Plattformübergreifend

Die Anwendung muss auf den gängigen Betriebssystemen (Windows, Linux, MacOS) funktionieren.

NFS02 - Single-Page Applikation

Die Anwendung darf während der Benutzung, nachdem der initiale Ladevorgang vollendet ist, keine Seite synchron laden.

NFS03 - Wartbarkeit

Die Anwendung muss klar strukturiert und dokumentiert sein. Änderungen am System müssen mit möglichst geringem Aufwand umsetzbar sein.

NFS04 - Skalierbarkeit

Die Anwendung muss skalierbar sein. Die Anwendung selbst darf deshalb keinen Zustand halten und muss eine beliebige Replizierbarkeit ermöglichen.

NFS05 - Offline

Die Anwendung soll auch bei Verbindungsverlust zum Server noch ordnungsgemäß funktionieren, insofern die Anwendung den initialen Ladevorgang erfolgreich abgeschlossen hat.

5 Konzeption

In diesem Kapitel wird das Konzept der Umsetzung vorgestellt und erläutert. Der Fokus dieser Thesis liegt auf der Entwicklung einer Anwendung zur dynamischen Visualisierung von zusätzlichen Informationen in BPMN 2.0 Prozessmodellen. Im ersten Unterkapitel wird die Architektur der Anwendung beschrieben. Daraufhin folgt im nächsten Abschnitt ein Überblick über die verwendeten Komponenten und es wird darauf eingegangen, wieso diese verwendet werden. Anschließend wird das Grundgerüst der Anwendung vorgestellt. Im letzten Unterkapitel werden die Sichten und die Funktionalitäten der Anwendung vorgestellt.

5.1 Architektur

Aufgrund dessen, dass die Anwendung sowohl plattformübergreifend als auch eine Single-Page Applikation sein muss (Anforderung: NFS01, NFS02), bietet sich eine Realisierung als Webanwendung an. Webanwendungen sind in jedem Betriebssystem mittels Browser aufrufbar, ohne dass es nötig wäre, die Anwendung zu installieren. Eine Single-Page Applikation bietet den Vorteil, dass die Ladezeiten für Serveranfragen wegfallen und somit der Eindruck entsteht, eine native Anwendung zu bedienen. Eine Webanwendung ist eine klassische Client-Server Architektur, die in Abbildung 5.1 dargestellt ist. Webbrowser wie Chrome, Safari und Firefox, repräsentieren den Client. Dieser schickt eine Anfrage an den Server, der daraufhin die angeforderten Ressourcen als Antwort an den Browser zurücksendet.

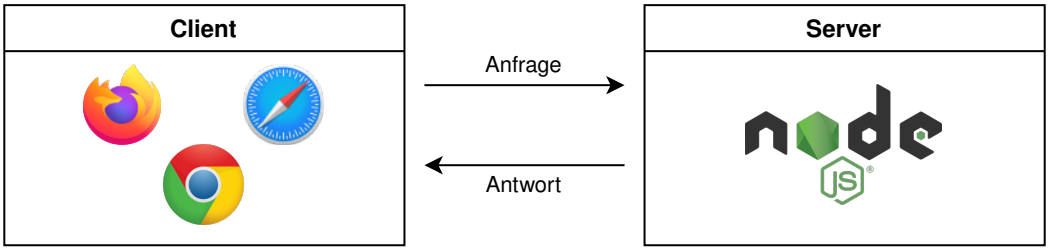

Abbildung 5.1: Client-Server Architektur. Abbildung basiert auf [51].

5.2 Komponentenbeschreibung

In diesem Abschnitt werden die einzelnen Komponenten, welche für die Konzeption relevant sind, aufgelistet. Des Weiteren erfolgt eine Begründung der Auswahl. Hierbei liegt der Fokus im ersten Abschnitt auf dem Server. Im anschließenden Abschnitt wird die Zusammensetzung der Webanwendung näher erläutert.

5.2.1 Server

Node.js ist als Webserver ausgewählt und wird für die Implementierung verwendet, da dieser ECMAScript 6 (ES6) unterstützt, mittels npm eine Paketverwaltung mitbringt und in vielen Anwendungen bereits verwendet wird [20]. Durch die Paketverwaltung gewinnt die Anwendung enorm an Flexibilität. Aufgrund dieser ist es möglich, bereits vorgefertigte Frameworks und Module, wie beispielsweise Bootstrap zur Anwendung hinzuzufügen.

5.2.2 Webanwendung

Die Webanwendung setzt sich aus mehreren Komponenten zusammen. Abbildung 5.2 gibt einen Überblick über die Zusammensetzung. Darauf folgt eine Erläuterung, wieso die jeweiligen Komponenten für die Realisierung zum Einsatz kommen.

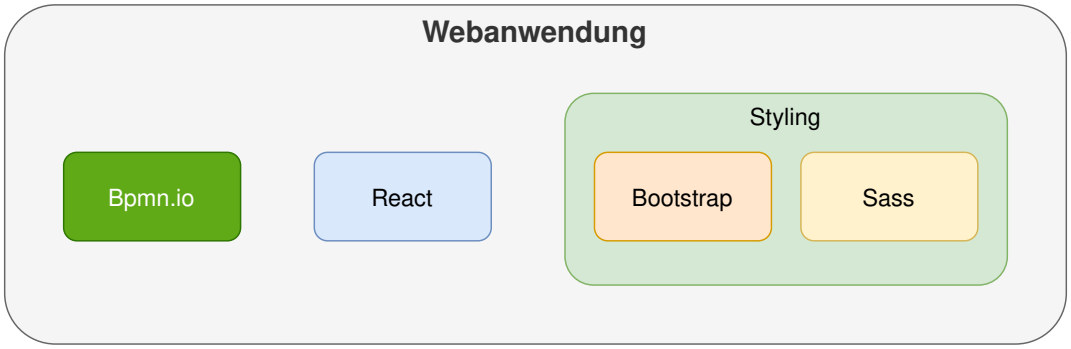

Abbildung 5.2: Komponentenübersicht der Webanwendung

Styling

Für das Styling der Anwendung wird Bootstrap (Abschnitt 2.5) und Sass (Abschnitt 2.4) eingesetzt. Bootstrap erleichtert das Styling durch vordefinierte Klassen und ermöglicht dadurch, den Fokus auf Funktionalitäten zu legen. Des Weiteren ist Bootstrap eines der bekanntesten CSS-Frameworks auf dem Markt. Dadurch sollten den Anwendenden das Aussehen und Verhalten der Elemente bekannt vorkommen. In Hinblick auf die Erwartungskonformität (Anforderung: NFU05 und Lernförderlichkeit NFU03) ist das ein klarer Vorteil. Für die Anwendungsfälle, die Bootstrap nicht abdeckt, wird Sass verwendet. Sass ist ausgewählt, da die Syntax nahe an klassischem CSS ist. Gleichzeitig erweitert es aber die Funktionalitäten von CSS beispielsweise durch Variabeln und der Möglichkeit, Elemente zu verschachteln. Des Weiteren ist Sass abwärtskompatibel mit sämtlichen CSS Versionen.

Bpmn.io

Für das Verarbeiten, Anzeigen und manipulieren der BPMN 2.0 Prozessmodelle wird bpmn-js, aus der Bpmn.io Bibliothek (Abschnitt 2.3), verwendet. Der Vorteil darin liegt, dass es kein Server-Backend benötigt. Des Weiteren ist es Open-Source und als npm Modul verfügbar. Außerdem ist es in JavaScript geschrieben und bietet hervorragende Möglichkeiten, die vorhandenen Funktionalitäten zu nutzen und zu erweitern. Generell sind alle Funktionalitäten der Webanwendung mit JavaScript implementiert.

React

Für die Benutzerschnittstelle selbst wird React (Abschnitt 2.6) verwendet. React erlaubt das Unterteilen der Benutzerschnittstelle in kleine Komponenten (Anforderung: NFS03). Die Aktualisierung der Komponenten übernimmt React selbst, sobald die gehaltenen Daten sich ändern. Dadurch muss keine manuelle Aktualisierung mittels JavaScript vorgenommen werden. Dieses Framework wird genutzt, um Single-Page Anwendungen zu erstellen, welches Teil der Anforderung an die Implementierung ist. Die Entwickler von React empfehlen zur Erstellung von Single-Page Applikationen die Verwendung von *create-react-app* [12]. Dieses Tool erstellt die Umgebung für die Implementierung und wird im folgenden Abschnitt näher betrachtet.

5.3 Grundgerüst

Das Grundgerüst der Architektur wird durch das Tool *create-react-app* erstellt. Es steht als Paket bereit und ist mittels npx Befehl ausführbar [11]. Bei der Ausführung erstellt dieses Ordner sowie Dateien und konfiguriert die Umgebung, sodass die Basis gelegt ist, um eine React Single-Page Applikation (Anforderung: NFS01, NFS02) zu realisieren [12]. Das Tool hat enormen Umfang und stellt viel Funktionalität zur Verfügung, daher werden ausschließlich die für die Arbeit relevanten Elemente näher betrachtet. Backend-Logik sowie eine Datenbank sind in diesem Tool nicht enthalten. Diese sind bei Bedarf frei wählbar [12], werden aber für die Implementierung dieser Arbeit nicht benötigt.

Create-react-app stellt eine Auswahl von vorkonfigurierten Tools zur Verfügung und installiert React mittels Abhängigkeit in der *package.json* Datei [11]. Wichtig für diese Arbeit sind die beiden Dateien: *index.html* sowie *index.js*. Diese dienen als Einstiegspunkt beim Starten der Anwendung und dürfen nicht unbenannt werden. Die Datei *index.html* enthält einen * *-container mit der ID *'root'*. Dieses Element dient als Container für die komplette React Anwendung. Die Datei *index.js* enthält eine Funktion, mit der das Wurzelelement der Anwendung in den zuvor genannten Container gerendert wird.

Die Anwendung wird per *npm start* Befehl gestartet. Dabei wird das mitgelieferte Startskript ausgeführt. Webpack beginnt von der *index.js* aus und parst durch die Anwendung, indem es jedem importierten Modul folgt. Daraus entsteht ein kompletter Abhängigkeitsgraph. Während des Parsens wird jede JavaScript Datei mittels Babel in eine JavaScript Version umgewandelt, die von mehr Browsern unterstützt wird und sich dort auch gleich verhält. Der Abhängigkeitsgraph wird von Webpack genutzt, um die einzelnen JavaScript Dateien und Module zu bündeln und diese dann als einzelne Datei in die *index.html* per *script* tag einzufügen. Der große Vorteil dabei ist, dass die komplette Anwendung mit einer Antwort vom Server zur Verfügung steht, ohne dass Inhalte synchron nachgeladen werden müssen (Anforderung: NFS04).

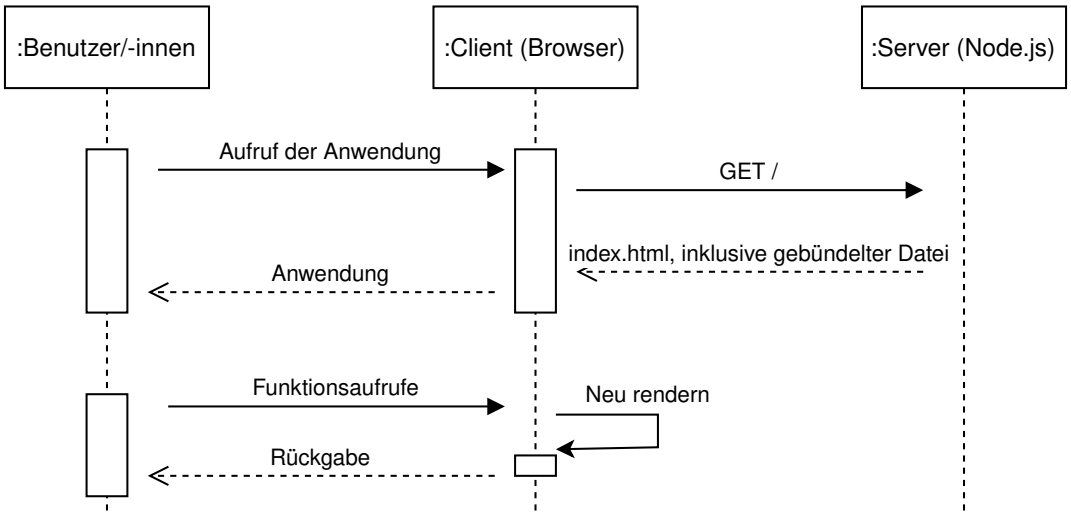

Abbildung 5.3: Sequenzdiagramm React Single-Page Applikation

Abbildung 5.3 zeigt das Zusammenspiel der beteiligten Objekte. Zu Beginn rufen die Benutzenden die Anwendung auf. Der Browser schickt daraufhin eine Anfrage an den Server. Der Server antwortet mit der *index.html* Datei, welche das gebündelte JavaScript enthält. Anschließend steht den Benutzenden die gerenderte Anwendung zur Verfügung.

Alle Funktionsaufrufe, die ab diesem Zeitpunkt folgen, werden ausschließlich clientseitig verarbeitet und lösen durch die Manipulation der Zustände ein neues Rendern der Komponenten aus (Anforderung: NFS04). Die neu gerenderten Komponenten stehen dann den Benutzenden wieder zur Verfügung. Weitere Aufrufe an den Ser-

ver sind nicht notwendig, da weder Backend noch Datenbank vorhanden ist. Die Benutzenden sind daher in der Lage, die Anwendung uneingeschränkt auch im Offline-Betrieb zu verwenden, ohne dass weitere Serveraufrufe notwendig wären (Anforderung: NFS05).

5.4 Mockups

In diesem Kapitel werden Mockups der Webanwendung präsentiert. Basierend auf diesen und den Anforderungen werden die Funktionalitäten genauer beschrieben und zentrale Bestandteile anhand von Diagrammen dargestellt. Die Anwendung soll lediglich aus zwei Sichten bestehen, die die Funktionalitäten weiter innerhalb unterteilen (Anforderung: NFU02). Folgend sind beide Sichten aufgelistet und beschrieben.

5.4.1 Startseite

Die Startseite wird angezeigt, sobald die Benutzenden den Server anfragen. Abbildung 5.4 zeigt ein Mockup der Startseite. Die Elemente sind mit roten Zahlen innerhalb runder Klammern markiert, um diese unmissverständlich zu referenzieren.

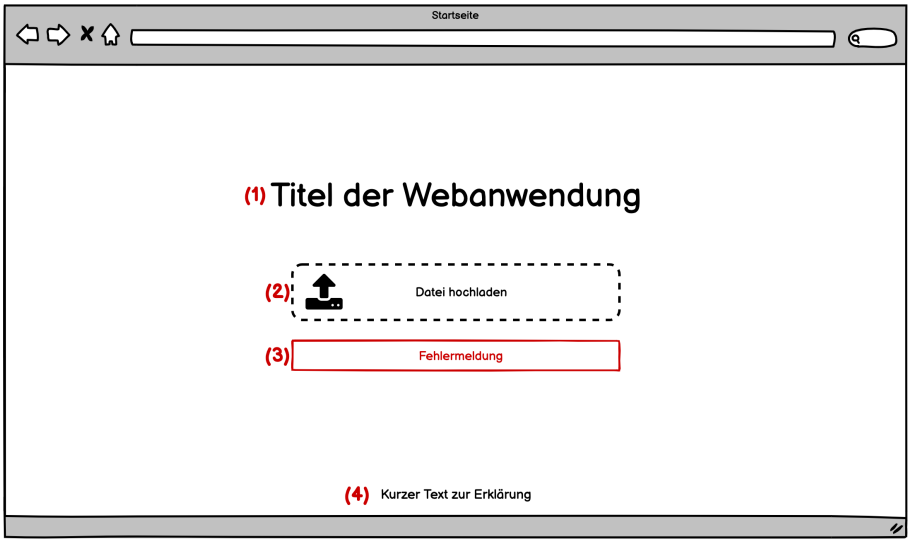

Abbildung 5.4: Mockup der Startseite

Element (1): Überschrift

Stellt den Namen der Anwendung als Überschrift dar. Dieser soll den Anwendenden zur Orientierung und als Wiedererkennungseffekt dienen.

Element (2): Datei hochladen

Ist ein für das Hochladen der BPMN 2.0 XML-Datei vorgesehenes Feld (Anforderung: F01). Das Hochladen beschränkt sich auf eine einzelne Datei. Mehrere Dateien werden nicht zugelassen. Element **(2)** soll zwei Möglichkeiten zur Verfügung stellen, um eine Datei auszuwählen. Abbildung 5.5 zeigt den Prozess des Hochladens der XML-Datei. Dieser Ablauf ist nachfolgend näher erläutert.

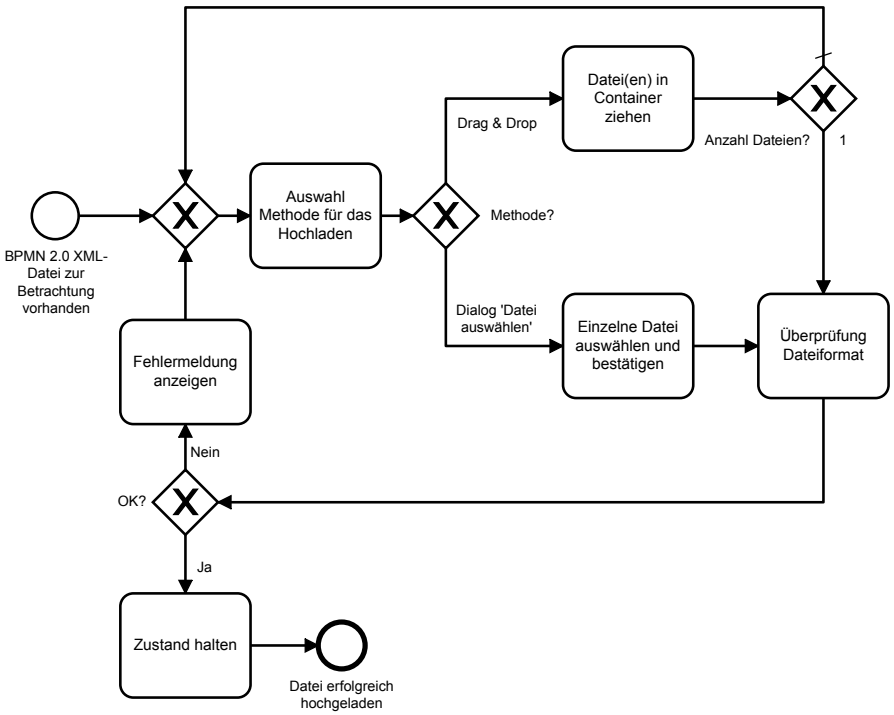

Abbildung 5.5: Ablauf ’Datei hochladen’ als BPMN 2.0 Prozessmodell

- **Drag-and-drop**

  Bei dieser Methode muss die BPMN 2.0 XML-Datei vorher markiert und anschließend mit gedrückter linker Maustaste in den gestrichelten Rahmen gezogen werden. Der Rahmen dient zur Abgrenzung und zeigt damit die Stelle, an der die Datei abgelegt werden darf. Sobald sich die Datei innerhalb der Markierung befindet, aber noch nicht abgelegt ist, erfolgt eine farbliche sowie textuelle Rückmeldung, über die Korrektheit der Anzahl der übergebenen Dateien (Anforderung: NFU04). Bei mehr als einer Datei färbt sich das Element rot und das Hochladen ist nicht möglich. Bei einer einzelnen Datei färbt sich das Element grün und das Ablegen ist erlaubt (Anforderung: NFU02).

- **Auswahl**

  Bei dieser Methode erfolgt ein Klick innerhalb der Markierung. Anschließend öffnet sich ein Betriebssystem spezifischer Dialog zur Dateiauswahl. Innerhalb des Dialoges haben die Benutzenden die Möglichkeit, eine einzelne Datei auszuwählen. Die Auswahl mehrerer Dateien ist nicht möglich. Erfolgt die Auswahl einer Datei, muss diese noch bestätigt werden.

Unabhängig von der gewählten Methode wird nach erfolgreicher Auswahl die Datei auf das korrekte Format hin überprüft (Anforderung: F02). Dabei wird die Datei zuerst auf die korrekte Dateiendung hin getestet und anschließend an Bpmn.io übergeben. Bpmn.io versucht die XML-Datei zu parsen und gibt das entsprechende Ergebnis zurück. Während der Überprüfung muss Element **(2)** unsichtbar und durch einen Ladekreis ersetzt werden. Das verhindert weitere Eingaben von Benutzenden und gibt zugleich Feedback, über einen stattfindenden Ladevorgang. Falls die Prüfung fehlschlägt, taucht sowohl Element **(2)** wieder auf, als auch eine Fehlermeldung (Element **(3)**). Nach erfolgreich abgeschlossener Prüfung wird das Dokument in einen Zustand gespeichert und die Sicht wechselt zum Hauptbildschirm.

Element (3): Fehlermeldung

Ist ein Container für Fehlermeldungen (Anforderung: F02). Der Hintergrund sowie die Textfarbe des Containers ist rot. Durch die Signalfarbe und der textuellen Beschreibung weist das Element die Benutzenden auf Fehler hin. Zu Beginn ist das

Element ausgeblendet, solange bis ein Fehler auftritt. Anwendende haben die Möglichkeit, diesen Container über ein 'X' auszublenden (Anforderung: NFU07). Die Fehlermeldung ist nicht zeitlich eingeschränkt und damit solange sichtbar, bis es manuell entfernt wird. Dadurch wird die Steuerbarkeit des Systems gewahrt und Anwendende haben ausreichend Zeit, die Fehlermeldung zu lesen.

Element (4): Erklärung

Gibt den Benutzenden eine kurze textuelle Erklärung, wofür die Anwendung verwendet werden kann (Anforderung: NFU02).

5.4.2 Hauptbildschirm

Der Hauptbildschirm wird angezeigt, sobald eine BPMN 2.0 XML-Datei auf der Startseite erfolgreich eingelesen und abgespeichert wird. Abbildung 6.7 zeigt ein Mockup des Hauptbildschirms. Die enthaltenen Elemente sind mit roten Zahlen innerhalb runder Klammern markiert, um diese unmissverständlich zu referenzieren.

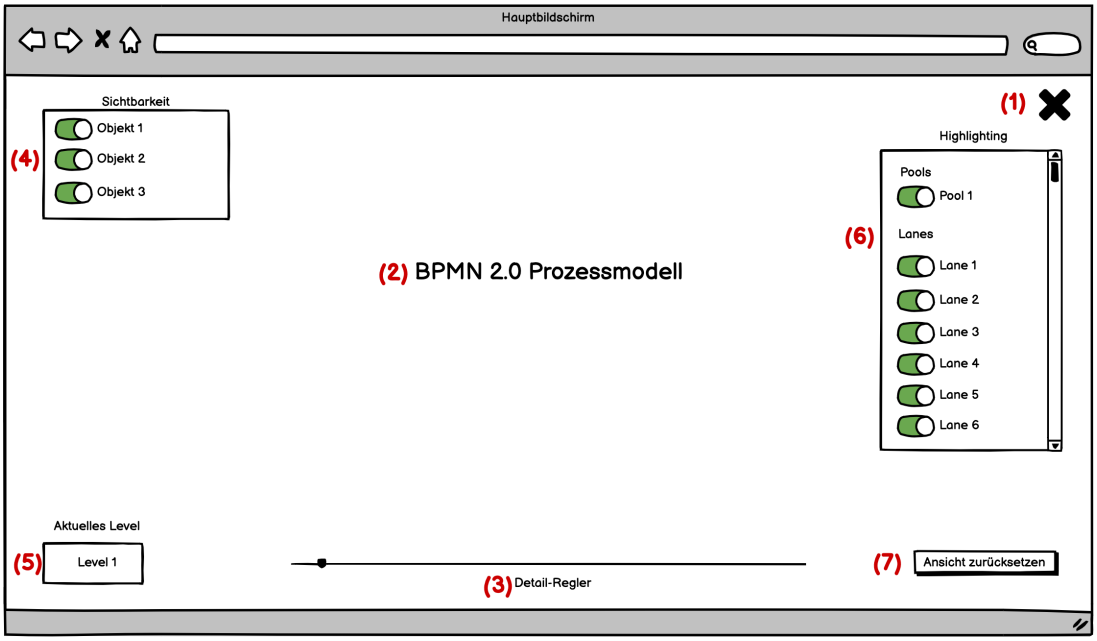

Abbildung 5.6: Mockup des Hauptbildschirms

Element (1): Zurückkehren

Wird durch ein 'X'-Symbol repräsentiert, welches permanent sichtbar ist. Durch ein Klick auf das Symbol gelangen die Anwendenden zurück zur Startseite, hierbei wird das aktuell dargestellte Prozessmodell verworfen.

Element (2): BPMN 2.0 Prozessmodell

Erstreckt sich über den gesamten sichtbaren Bereich des Browserfensters. Damit soll der zur Verfügung stehende Platz bestmöglich ausgereizt werden. Alle anderen Elemente überlagern dieses. Das Element dient zur Darstellung der hochgeladenen BPMN 2.0 XML-Datei als Prozessmodell. Sobald die Sicht auf den Hauptbildschirm wechselt, übernimmt das Framework Bpmn.io die Datei, parst die XML-Datei und rendert das Prozessmodell (Anforderung: F03). Nach dem rendern wird die Ansicht zentriert und der Zoom so eingestellt, dass das komplette Modell sichtbar ist. Anschließend wird das navigieren innerhalb des Modells und das Zoomen gewährleistet. Dafür stellt Bpmn.io den bereits erwähnten *NavigatedViewer* zur Verfügung (Anforderung: F07). Alle Manipulationen am Prozessmodell, die dynamisch während der Betrachtung vorgenommen werden, erfolgen direkt über Bpmn.io und werden unmittelbar danach visualisiert. An der Stelle sei nochmals erwähnt, dass sämtliche Interaktionen mit Bpmn.io ausschließlich lokal im Browser stattfinden.

Element (3): Detail-Regler

Um die Detailstufe zu ändern, wird ein sogenannter Range-Slider (Element **(3)**) verwendet. Technisch ist der Range-Slider ein *<input>*-Element, welches sein Aussehen über Bootstrap bekommt. Der Regler ist nicht stufenlos verstellbar, sondern nur in den insgesamt vier definierten Positionen (Anforderung: F05). Somit hat jedes Level eine gesonderte einrastbare Position. Initial ist Stufe drei eingestellt, die das Prozessmodell unverändert anzeigt. Ein Range-Slider eignet sich für diese Aufgabe, da die einstellbaren Werte fest vorgegeben werden können und sich die Anwendenden anhand der Position der Anzeige ableiten können, in welcher Stufe sie sich in etwa befinden. Ein Stufenwechsel löst eine Manipulation des Prozessmodells und eine Veränderung des Zustands der Elemente **(4)** und **(5)** aus. Details zu

diesen Elementen sind in den entsprechenden Abschnitten näher erläutert. Folgende Auflistung zeigt auf, welche Änderungen am Prozessmodell in den unterschiedlichen Stufen vorgenommen werden (Anforderung: F04). Aufgrund der Tatsache, dass mehrere Stufen bei einem Wechsel übersprungen werden können, muss in jeder Stufe die Sichtbarkeit sämtlicher Elemente entsprechend eingestellt werden.

- **Stufe 1:** Datenobjekte, Datenspeicher inklusive zugehöriger Assoziationen, Erklärungen und Annotationen vom Prozessmodell entfernen.
- **Stufe 2:** Datenobjekte, Datenspeicher inklusive zugehöriger Assoziationen dem Prozessmodell hinzufügen. Erklärungen und Annotationen vom Prozessmodell entfernen.
- **Stufe 3:** Prozessmodell darstellen ohne Veränderung. Erklärungen vom Prozessmodell entfernen.
- **Stufe 4:** Prozessmodell darstellen ohne Veränderung. Erklärungen dem Prozessmodell hinzufügen (Anforderung: F08).

Element (4): Objekt Sichtbarkeit

Ist ein Bedienelement, das die Sichtbarkeit einzelner BPMN 2.0 Elemente anzeigt und steuert. In Anforderung F06 und F08 werden die Elemente erläutert, welche aufgelistet werden sollen. Die Sichtbarkeit ist mittels sogenannter *switches* veränderbar, hierbei handelt es sich um *<input>*-Elemente vom Typ *checkbox*. Diese zeigen den aktuellen Zustand und sind intuitiv zu bedienen. Bei einem klick auf die *switches* oder den Elementnamen wird das entsprechende BPMN 2.0 Element ein- bzw. ausgeblendet. Diese Feinsteuerung beeinflusst das Detaillevel nicht. Verändert Element **(3)** die Detailstufe, wird auch die Sichtbarkeit der gelisteten BPMN 2.0 Elemente an die Stufe angepasst (Anforderung: NFU06).

Die Sichtbarkeit von den Erklärungen, die an die BPMN 2.0 Elemente angeheftet werden, ist in diesem Container einstellbar. Mit der Erklärung ist die Bennenung der BPMN 2.0 Elemente gemeint. Ein Konzept zur möglichen Darstellung der Erklärungen ist in Abbildung 5.7 zu sehen.

Die Erklärungen werden durch örtliche Nähe zu den Elementen und mittels farblicher Abhebung visualisiert. Bpmn.io bietet dafür die Möglichkeit an, das Prozess-

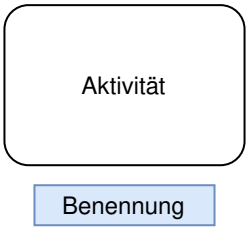

Abbildung 5.7: Konzept einer Erklärung in Form von Benennung

modell mittels Overlays zu erweitern und damit die Erklärung umzusetzen. Die Benennungen, die als Erklärungen dienen, werden automatisiert aus Bpmn.io extrahiert und an jedes erste Element angebracht. Wenn beispielsweise 20 Aktivitäten im Prozess sind, darf nur bei der ersten Aktivität die Erklärung erscheinen, um die Lesbarkeit nicht zu verringern.

Die Sichtbarkeit von Nachrichtenflüssen ist Bestandteil dieses Containers, wird aber nicht über den Detail-Regler beeinflusst. Nachrichtenflüsse sind unter Umständen für den Sequenzfluss essenziell. Beispielsweise wenn ein eingehender Nachrichtfluss ein Prozess startet. Bei Nachrichtenflüssen besteht dennoch die Gefahr, dass diese andere Elemente überlagern und somit die Lesbarkeit verringern. Deshalb werden diese ausblendbar sein, jedoch ausschließlich manuell über dieses Objekt.

Element (5): Aktuelles Level

Ist eine Anzeige, die lediglich die aktuelle Detailstufe, in der sich das Prozessmodell befindet, durch farbliche Markierung anzeigt. Alle weiteren Stufen sind ebenso Bestandteil und mit einem zusammenfassenden Begriff aufgelistet (Anforderung: F09). Diese Anzeige dient hauptsächlich zur Orientierung.

Element (6): Highlighting

Ist ein Bedienelement, das sämtliche Pools und Lanes voneinander separiert auflistet (Anforderung: F10). Eine Auflistung erfolgt durch den Namen der Elemente. Außerdem erhält jedes Element ein *switch*. Falls einem Swimlane-Element keinen Namen zugewiesen ist, wird der Identifikator des Elements verwendet. Wenn die Anzahl der dargestellten Elemente die Höhe des Bildschirms übersteigt, stellt der

Container Scrollbalken zur Verfügung. Somit ist gewährleistet, dass alle enthaltenen Elemente in der Liste auswählbar sind. Die *switches* zeigen, ob der jeweilige Pool oder die Lane innerhalb des Prozessmodells farblich markiert ist. Des Weiteren lassen sich die Swimlanes darüber markieren (Anforderung: F11). Verschachtelte Swimlanes werden durch Veränderung der Transparenz voneinander abgehoben. Abbildung 5.8 zeigt eine mögliche farbliche Markierung von geschachtelten Swimlanes.

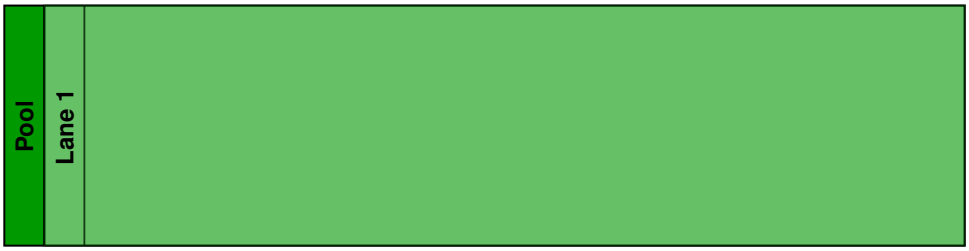

Abbildung 5.8: Konzept der farblichen Markierung von Swimlanes

Element (7): Ansicht zurücksetzen

Ist ein Button, der die Ansicht des Prozessmodells zurücksetzt (Anforderung: NFU0 2). Das Zurücksetzen der Ansicht zentriert das Modell und der Zoomfaktor wird so verändert, dass das komplette Prozessmodell sichtbar ist. Die Funktion dieses Buttons wird bei Bedarf per Tooltipp angezeigt (Anforderung: NFU03). Der Button ist zwar nicht zwingend erforderlich, erhöht aber die Usability. Insbesondere verbessert dieser die Fehlertoleranz (Anforderung: NFU04) und Steuerbarkeit (Anforderung: NFU07) der Anwendung.

5.4.3 Zusammenspiel

In diesem Abschnitt wird das Zusammenspiel der Sichten aufgezeigt und ein Ablauf aus Systemperspektive beschrieben (Anforderung: NFU01). Abbildung 5.9 zeigt diesen Ablauf, der nachfolgend näher beschrieben wird. Details zu den einzelnen Schritten sind den vorangegangenen Abschnitten zu entnehmen.

Der Prozess beginnt, sobald die Anwendenden die Webanwendung öffnen. Dabei wird die Sicht 'Startseite' angezeigt. Sobald eine Datei erfolgreich hochgeladen ist,

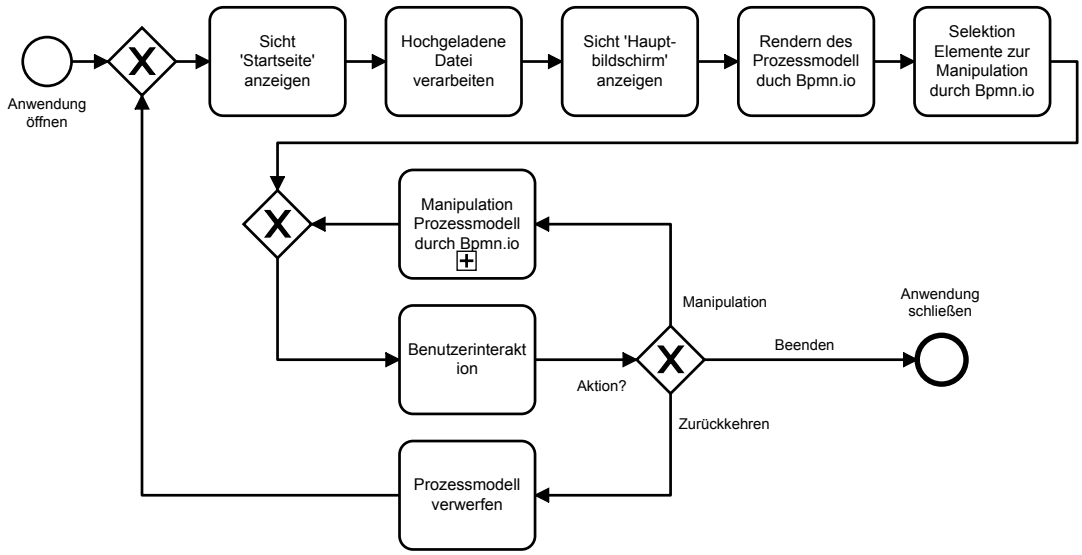

Abbildung 5.9: Ablauf der Anwendung als BPMN 2.0 Prozessmodell aus Systemperspektive

erfolgt die Verarbeitung. Im Anschluss daran wechselt die Sicht auf den ’Hauptbildschirm’. Nach dem Sichtwechsel bekommt das Framework Bpmn.io diese Daten und rendert damit das Prozessmodell. Im Anschluss werden alle Elemente, mit denen der Benutzer interagieren kann über Bpmn.io im Vorfeld selektiert. Ab diesem Zeitpunkt steht den Anwendenden das Prozessmodell zur Betrachtung bereit und eine Interaktion ist möglich. Die Auswahl von *’Zurückkehren’*, verwirft das aktuelle Prozessmodell und die Sicht ’Startseite’ wird wieder angezeigt. Die Auswahl von *’Manipulation’*, wie beispielsweise navigieren oder die Detailstufe verstellen wird über das Framework Bpmn.io gesteuert. Anwendende haben auch die Möglichkeit, die Anwendung zu schließen und damit die Betrachtung zu beenden.

6 Implementierung

In diesem Kapitel werden lediglich ausgewählte Aspekte der Implementierung vorgestellt. Diese repräsentieren somit nicht die komplette Anwendung. Sämtliche Anleitungen und Texte der Benutzeroberfläche sind auf Englisch und deshalb auch die Abbildungen, die die jeweiligen Komponenten zeigen. Englisch ist die meist gesprochene Sprache und damit soll das Endprodukt möglichst vielen Anwendenden zur Verfügung stehen.

Im ersten Abschnitt wird das Hochladen der Datei betrachtet. Mittelpunkt des folgenden Abschnitts ist die Initialisierung des Frameworks Bpmn.io und die Selektion der Elemente, die für die Manipulation des Prozessmodells benötigt werden. Anschließend wird auf die Auflistung und Hervorhebung der Swimlanes näher eingegangen. Während in Abschnitt vier die Implementierung von Overlays aufgezeigt werden. Im letzten Abschnitt wird darauf eingegangen, wie die Veränderung der Sichtbarkeit der BPMN 2.0 Elemente implementiert ist.

6.1 Datei hochladen

Um die BPMN 2.0 XML-Datei gemäß den Anforderungen hochzuladen, wird react-dropzone [18] verwendet. Das ist ein externes npm-Modul, welches mit React implementiert und HTML5 konform ist. Dieses Modul stellt Funktionen bereit, um Dateien entweder per Drag & Drop oder per Auswahlmenü hochzuladen.

Abbildung 6.1 visualisiert die Komponente für den Upload der Datei. In der Mitte der Komponente befindet sich eine knappe Instruktion, die ungeübten Benutzern den Einstieg erleichtern soll. Des Weiteren ist unten auf der Abbildung das Element zur Darstellung der Fehlermeldung sichtbar.

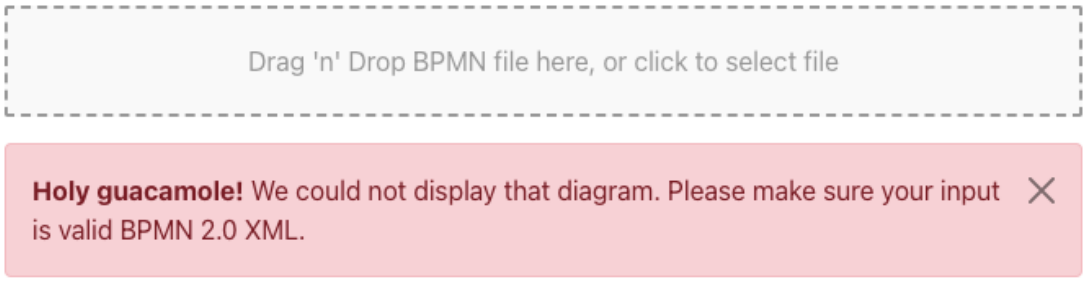

Abbildung 6.1: Komponente für das Hochladen von BPMN 2.0 XML-Dateien, inklusive eingeblendeter Fehlermeldung

Listing 6.1 zeigt ab Zeile 1 die Funktion *onDrop*. Diese wird aufgerufen, sobald die Datei ausgewählt oder innerhalb der Markierung abgelegt wird. Als Übergabeparameter erwartet die Funktion ein Array mit allen Dateien, die den vorgegebenen Kriterien entsprechen. In Zeile 2 erfolgt eine Iteration über das Datei-Array. Mit der Zuweisung in der darauffolgenden Zeile wird den Benutzenden ein Ladesymbol angezeigt. Das hat außerdem noch den Effekt, dass die Komponente für das Hochladen der Datei ausgeblendet wird um weitere Benutzereingaben zu verhindern. Danach erfolgt eine Instanziierung des *FileReader*s, mit dem die Dateien gelesen werden. Anschließend wird die Datei als Text von der *FileReader* Instanz verarbeitet. Die Funktion in Zeile 7 wird aufgerufen, sobald das Lesen der Datei erfolgreich beendet ist. Zuerst erfolgt eine Prüfung auf die korrekte Dateiendung (Zeile 8). Bei erfolgreicher Prüfung, versucht Bpmn.io die Datei als XML zu importieren (Zeile 10). Ist das erfolgreich, wird die Datei als Zustand gespeichert (Zeile 12) und die Sicht wechselt auf den Hauptbildschirm. Ist die Dateiendung nicht korrekt oder geht bei der Importierung etwas schief, wird eine Fehlermeldung angezeigt (Zeile 15, 18). Des Weiteren ist der Bereich für den Dateiupload wieder sichtbar (Zeile 21). In der letzten Zeile wird dem importierten Modul die *onDrop* Funktion und die Einstellungen für das hochladen der Datei übergeben. Dabei ist das Hochladen maximal einer Datei möglich, wie in den Anforderungen angegeben. Außerdem werden Konstanten definiert, die für die Veränderung der Benutzeroberfläche relevant sind. Beispielsweise wird mit der Konstante *isDragActive* der Text im Bereich zum ablegen der Dateien verändert sobald eine Datei in den Bereich gezogen wird.

```
const onDrop = useCallback((acceptedFiles) => {
    acceptedFiles.forEach((file) => {
        setShowLoading(true);
        const reader = new FileReader();
        reader.readAsText(file);

        reader.onload = () => {
            if (file.name.split(".").pop().toLowerCase() === "
                bpmn") {
                let view = new Viewer();
                view.importXML(reader.result)
                    .then(()=>{
                        setFileData(reader.result);
                    },
                        (onReject)=>{
                            setShowAlert(true);
                        });
            } else {
                setShowAlert(true);
            }
        };
        setShowLoading(false);
    });
}, []);

const {getRootProps, getInputProps, isDragActive, isDragAccept}
     = useDropzone({onDrop, maxFiles: 1, multiple: false});
```

Listing 6.1: Funktion zum Hochladen der Datei

Die Komponente für das Hochladen der BPMN 2.0 XML-Datei befindet sich auf dem Startbildschirm, welcher in Abbildung 6.2 zu sehen ist. Oben steht der Titel der Anwendung. Während sich unten eine knappe textuelle Anleitung befindet. Die Upload Komponente ist mit roter Farbe hinterlegt und zeigt an, dass sich zu viele Dateien im Container befinden. Diese Aktion wird ausgelöst, sobald mehr als eine Datei in den Bereich gezogen wird. Der Originalzustand der Komponente ist in Abbildung 6.1 zu sehen. Die Funktion in Listing 6.1 wird ausgeführt, sobald ein*e Benutzer*in eine einzelne Datei innerhalb des Bereichs ablegt.

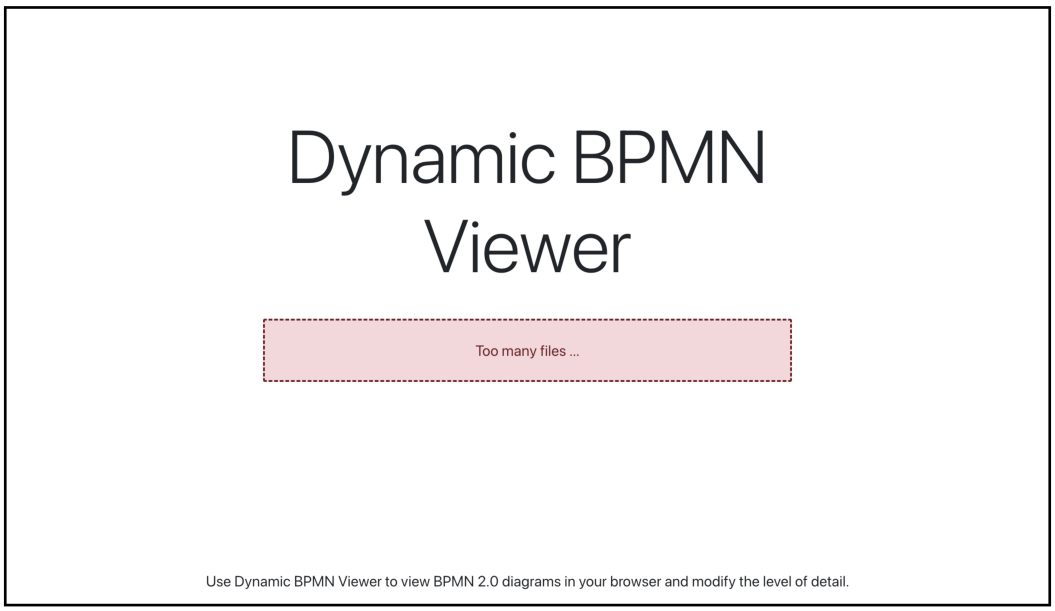

Abbildung 6.2: Sicht auf den Startbildschirm der Anwendung

6.2 Initialisierung und BPMN Element Selektion

Sobald die Datei erfolgreich eingelesen ist, erfolgt der Sichtwechsel zum Hauptbildschirm. Auf dem Hauptbildschirm muss das Framework Bpmn.io zur Anzeige und Verarbeitung des Prozessmodells initialisiert werden. Listing 6.2 zeigt diese Initialisierung. Die Funktion in Zeile 1 wird aufgerufen, sobald das Rendern des Hauptbildschirms abgeschlossen ist. Die Funktion *useEffect* wiederum ruft die Funktion in Zeile 5 auf. Innerhalb dieser Funktion wird der *NavigatedViewer* aus Bpmn.io instanziiert und per ID in die Benutzeroberfläche eingesetzt (Zeile 6). In Zeile 11 importiert der *NavigatedViewer* die eingelesene Datei. Sobald das erfolgreich abgeschlossen ist, werden die Komponenten aus dem Framework Bpmn.io als Konstanten gespeichert (Zeile 12-15), die anschließend für die Selektion und Manipulation des Prozessmodells benötigt werden. Ab Zeile 17 ist zu sehen, wie die BPMN 2.0 Elemente selektiert und gespeichert werden. Im Anschluss an folgendes Listing, wird die Selektion näher betrachtet. Die Funktion in der letzten Zeile zentriert das Prozessmodell und stellt das Zoomlevel so ein, dass das Prozessmodell komplett zu sehen ist.

```
useEffect(() => {
    setupModeler();
}, []);

const setupModeler = () => {
    viewer = new NavigatedViewer({ container: "#canvas" });
    importXML(fileData);
};

const importXML = (fileData) => {
    viewer.importXML(fileData).then(() => {
        setCanvas(viewer.get("canvas"));
        setOverlays(viewer.get("overlays"));
        setElementRegistry(viewer.get("elementRegistry"));
        setGraphicsFactory(viewer.get("graphicsFactory"));

        setPresentFirstElements(selectAllFirstElems());
        setAllPools(selectElements("Participant"));
        setAllLanes(selectElements("Lane"));
        setAllAnnotations(selectElements("TextAnnotation"));
        setAllDataObjects(selectElements("DataObjectReference")
            );
        setAllDataStores(selectElements("DataStoreReference"));
        setAllMessageFlows(selectElements("MessageFlow"));
        //Center and zoom to display whole bpmn model
        viewer.get("canvas").zoom("fit-viewport", "auto");
    });
};
```

Listing 6.2: Initialisierung von Bpmn.io und Selektion der BPMN 2.0 Elemente

Listing 6.3 zeigt die Selektion der BPMN 2.0 Elemente. Die Funktion *selectElements* erwartet den Namen der BPMN 2.0 Elemente, die selektiert werden sollen (Zeile 1). In der *elementRegistry* aus dem *NavigatedViewer* befinden sich alle Elemente, die Bestandteil des gerade betrachtenden Prozessmodells sind. Auf dieses Array wird eine filter Funktion angewandt, mit der Elemente anhand ihres Typs selektiert und als Array zurückgegeben werden (Zeile 2). Der Typ setzt sich aus dem Präfix *'bpmn:'* und dem eigentlichen Elementnamen zusammen. Zur Überprüfung, ob das Element den gewünschten Typ hat, wird die Funktion *is* aus Bpmn.io genutzt (Zeile 3). Diese gibt *true* zurück, wenn der Typ übereinstimmt und *false*, falls nicht.

Die Funktion *selectAllFirstElems* (Zeile 7) aus Listing 6.3 dient dazu, alle Elemente, die zum ersten Mal in dem Prozessmodell auftauchen, auszuwählen. Zeile 8 zeigt die Selektion aller BPMN 2.0 Elemente im Prozessmodell und die Zuweisung des resultierenden Arrays. Anschließend wird in Zeile 12 über das Array iteriert und überprüft ob dieser Element-Typ bereits in dem Rückgabearray enthalten ist (Zeile 15). Wenn sich noch kein Element des Typs in dem Rückgabearray befindet und das Element auch nicht das Prozessmodell selbst ist (Zeile 19), wird es hinzugefügt (Zeile 20). Nach der Iteration erfolgt die Rückgabe des Arrays (Zeile 23). In diesem Array ist immer jeweils das erste auftauchende Element eines Typs enthalten.

```
1  const selectElements = (bpmnElementName) => {
2      return viewer.get("elementRegistry").filter((element) => {
3          return is(element, "bpmn:" + bpmnElementName);
4      });
5  };
6
7  const selectAllFirstElems = () => {
8      let allElements = viewer
9          .get("elementRegistry")
10         .filter((element) => element.type.startsWith("bpmn:"));
11     let firstElements = [];
12     allElements.forEach((element) => {
13         let alreadyAdded = false;
14         firstElements.forEach((elem) => {
15             if (elem.type === element.type) {
16                 alreadyAdded = true;
17             }
18         });
19         if (!alreadyAdded && element.type !== "bpmn:
               Collaboration") {
20             firstElements.push(element);
21         }
22     });
23     return firstElements;
24 };
```

Listing 6.3: Selektion spezifischer BPMN 2.0 Elemente

6.3 Auflistung und Hervorhebung der Swimlanes

Um die Pools sowie Lanes aufzulisten und bei Bedarf farblich hervorzuheben, werden die entsprechenden Arrays (*allPools*, *allLanes*) genutzt, die bei der Initialisierung in Abschnitt 6.2 entstanden sind. Abbildung 6.3 zeigt die Komponente zur Auflistung der Swimlanes. Auf der linken Seite ist ein Prozessmodell zu sehen. Bei diesem ist der Pool sowie die obere Lane farblich markiert.

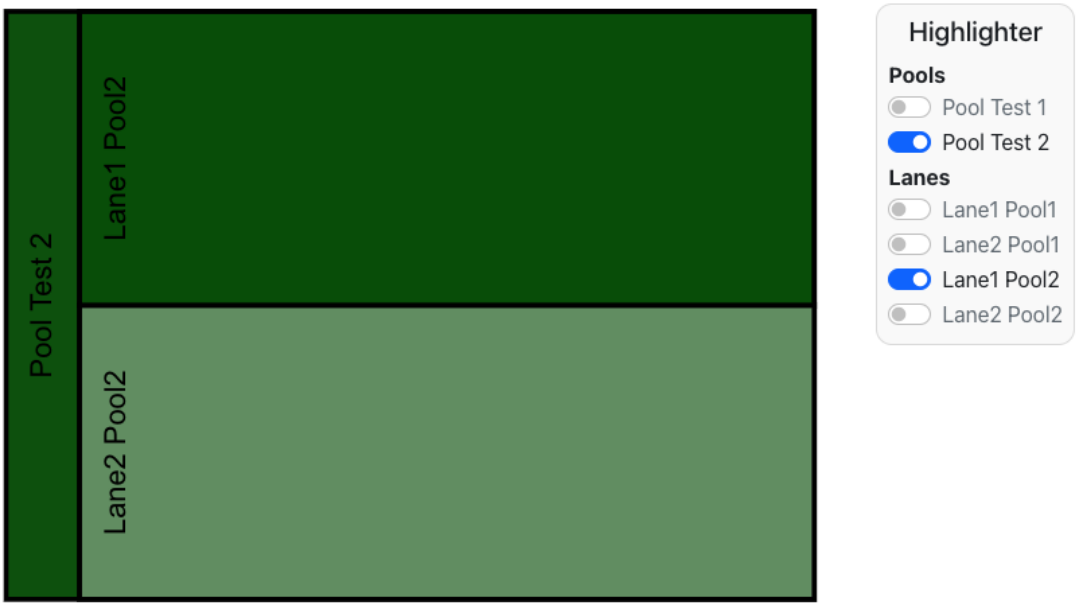

Abbildung 6.3: Komponente für die Auflistung und farbliche Markierung von Swimlanes, inklusive Beispiel Prozessmodell

Listing 6.4 zeigt die Auflistung der Pools. Diese Implementierung ist analog zu den Lanes. Zeile 1 zeigt die Iteration über sämtliche Pools. Der angezeigte Name ist dabei entweder der zugewiesene Name, falls dieser existiert oder die generierte ID (Zeile 2). Anschließend bekommt eine React Komponente die Funktionen für die Markierung übergeben und rendert den Namen inklusive *switch* Button (Zeile 6-15).

```
1  {allPools.map((pool) => {
2      let name =
3          pool.businessObject.name == null
4              ? pool.id
5              : pool.businessObject.name;
6      return (
7          <HighlightToggle
8              elem={pool}
9              label={name}
10             highlightedElements={highlightedElements}
11             setHighlightedElements={setHighlightedElements}
12             highlightElement={highlightElement}
13             removeHighlightElement={removeHighlightElement}
14         />
15     );
16 })}
```

Listing 6.4: Rendern der Pools

Listing 6.5 zeigt die beiden Funktionen, die das Markieren der Swimlanes übernehmen. Die Funktion in Zeile 1 färbt das übergebene Element Grün (Zeile 2). Für die Umsetzung der Färbung stellt Bpmn.io die *graphicsfactory* zur Verfügung (Zeile 5). Außerdem sorgt diese dafür, dass die geschachtelten Elemente, wie beispielsweise Lanes innerhalb eines Pools, eine transparentere Abstufung der Farbe enthalten. Die Funktion ist generisch implementiert, dadurch ist es sogar möglich, nicht nur Swimlanes, sondern alle Elemente und Verbindungen zu markieren. Analog zu dieser Funktion färbt *removeHighlightElement* den Hintergrund auf die Standard Farbe Weiß (Zeile 9).

```
const highlightElement = (elem) => {
    elem.businessObject.di.set("fill", "rgba(0, 80, 0, 1)");
    const gfx = elementRegistry.getGraphics(elem);
    const type = elem.waypoints ? "connection" : "shape";
    graphicsFactory.update(type, elem, gfx);
};

const removeHighlightElement = (elem) => {
    elem.businessObject.di.set("fill", "rgba(255, 255, 255, 1)"
        );
    const gfx = elementRegistry.getGraphics(elem);
    const type = elem.waypoints ? "connection" : "shape";
    graphicsFactory.update(type, elem, gfx);
};
```

Listing 6.5: Markierung der Swimlanes

6.4 Overlays

Für die Benennung der Elemente per Overlay muss eine geeignete Positionierung dieser zusätzlichen Information festgelegt werden. Abbildung 6.4 zeigt ein Prozessmodell mit den implementierten Benennungen per Overlay. Dabei sind diese Overlays blau gefärbt, um sich vom Prozessmodell abzuheben und je nach Elementtyp positioniert.

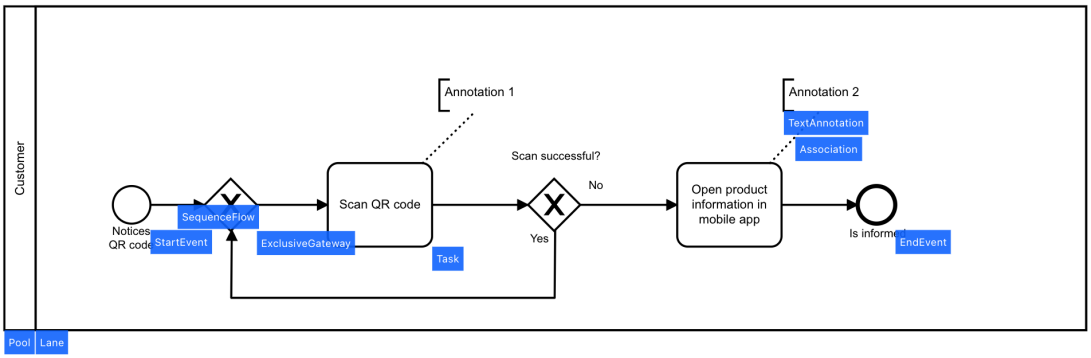

Abbildung 6.4: Darstellung der Overlays im Prozessmodell

Diese Positionierung übernehmen die beiden Hilfsfunktionen, welche in Listing 6.6 zu sehen sind. Die Funktion in Zeile 1 berechnet den Mittelpunkt zweier Punkte, um das Overlay genau in der Mitte eines Verbindungselements, beispielsweise einer Assoziation, anzubringen. Während die Funktion in Zeile 7 ein Objekt zur Positionierung zurückgibt, abhängig vom Element-Typ (Zeile 14-24). Die verschiedenen Anbringungspunkte der Overlays sorgen dafür, dass sich die Overlays so nahe wie möglich am Element selbst befinden und möglichst wenig davon verdecken.

```
1  static calculateCenterOfTwoPoints = (element) => {
2      let a = element.waypoints[0];
3      let b = element.waypoints[1];
4      return { top: Math.abs(a.y - b.y) / 2, left: Math.abs(a.x -
           b.x) / 2 };
5  };
6
7  static getPosition = (element, type) => {
8      let downLeft = { bottom: 0, left: 0 };
9      let downRight = { bottom: 0, right: 0 };
10
11     if (element.waypoints) {
12         return this.calculateCenterOfTwoPoints(element);
13     }
14     switch (type) {
15         case "Pool":
16         case "Lane":
17         case "TextAnnotation":
18             return downLeft;
19         case "StartEvent":
20         case "EndEvent":
21             return { right: 0, bottom: -6 };
22         default:
23             return downRight;
24     }
25 };
```

Listing 6.6: Positionierung der Overlays

Listing 6.7 das Hinzufügen und Entfernen der Overlays. Die Funktion in Zeile 1 bekommt alle Elemente übergeben, die ein Overlay erhalten sollen. Über sämtliche Elemente wird iteriert (Zeile 2). Dabei wird der Typ des Elements herausgezogen

(Zeile 4) und die Funktion in Zeile 11 aufgerufen. In dieser Funktion wird die Position des Elements festgelegt (Zeile 12) und das Overlay mit der Benennung initialisiert (Zeile 13). Das Overlay muss zwingend ein HTML-Element sein. Anschließend erfolgt die Übergabe des Overlays an das Framework, um dieses zu rendern (Zeile 14). Bpmn.io stellt des Weiteren eine Funktion namens *clear* zur Verfügung, die auf Overlay-Elemente angewandt wird. Mit dieser werden sämtliche Overlays entfernt (Zeile 18).

```
1  const addOverlays = (elements) => {
2      elements.forEach((elem) => {
3          let content =
4              elem.type.split(":")[1] === "Participant"
5                  ? "Pool"
6                  : elem.type.split(":")[1];
7          addOverlay(elem, content);
8      });
9  };
10
11 const addOverlay = (element, content) => {
12     let position = BdvUtil.getPosition(element, content);
13     let html = '<div class="diagram-note p-1">' + content + "<
           div/>";
14     return overlays.add(element, { position: position, html:
           html });
15 };
16
17 const removeOverlays = () => {
18     overlays.clear();
19 };
```

Listing 6.7: Hinzufügen und Entfernen der Overlays

6.5 Sichtbarkeit der BPMN 2.0 Elemente

Bei der Einstellung der Detailstufe durch den Regler oder durch die Feineinstellung mittels *switches* werden die entsprechenden Elemente dem Prozessmodell hinzugefügt oder entfernt. Eine Abbildung der Komponente zur Feinsteuerung ist in Abbildung 6.5 zu sehen. Alle sichtbaren Elemente sind mit blauen *switches* hinterlegt.

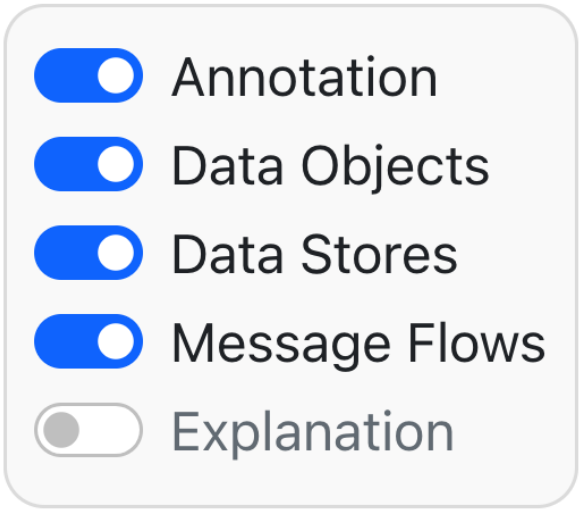

Abbildung 6.5: Feinsteuerung der Sichtbarkeit einzelner Elemente

Für das Hinzufügen von Elementen sind grundlegende Funktionen implementiert, welche in Listing 6.8 zu sehen sind. Das Entfernen der Elemente ist von der Vorgehensweise analog und wird somit nicht näher betrachtet. Die Funktion in Zeile 1 bekommt ein Array von Elementen übergeben, über das iteriert wird. Dabei wird mittel ID überprüft, ob das Element bereits im Prozessmodell vorhanden ist (Zeile 3). Falls nicht, wird es hinzugefügt und sämtliche zugehörige eingehenden sowie ausgehenden Verbindungen ebenso. Die Funktion *addConnections* in Zeile 10 separiert die Verbindungen (Zeile 11 und 12) und ruft jeweils *addConnectionArray* (Zeile 14-15) auf. Diese Funktion (Zeile 18) wiederum iteriert über alle Verbindungen (Zeile 19) und fügt diese hinzu (Zeile 24), falls sie nicht bereits vorhanden sind. Ein Label wird vom Framework Bpmn.io als Verbindung gesehen und muss gesondert in Zeile 22 als *Shape* hinzugefügt werden.

```
const addElements = (elements) => {
    elements.forEach((elem, index) => {
        if (!elementRegistry.get(elem.id)) {
            addConnections(elem);
            canvas.addShape(elem);
        }
    });
};

const addConnections = (element) => {
    let incomingCons = element.incoming;
    let outgoingCons = element.outgoing;

    addConnectionArray(incomingCons);
    addConnectionArray(outgoingCons);
};

const addConnectionArray = (connectionArray) => {
    connectionArray.forEach((con, index) => {
        if (!elementRegistry.get(con.id)) {
            if (con.type === "label") {
                canvas.addShape(con);
            } else {
                canvas.addConnection(con);
            }
        }
    });
};
```

Listing 6.8: Hinzufügen von Elementen und Verbindungen zum Prozessmodell

Das Verändern der Stufe des Reglers, löst die Funktion in Listing 6.9 aus. Die Oberfläche des Reglers ist in Abbildung 6.6 dargestellt.

Abbildung 6.6: Regler zur Steuerung der Detailstufe

Die Funktion *changeDetailLevel* in Zeile 1 von Listing 6.9 bekommt die eingestellte Stufe übergeben. Diese wird als Zustand gespeichert (Zeile 2). Die *switch* Anweisung (Zeile 3-32) bildet die vier vordefinierten Detailstufen ab. Je nach Stufe wird die Sichtbarkeit der Elemente eingestellt. Die Funktionsaufrufe innerhalb der unterschiedlichen Fallunterscheidungen nehmen einen boolschen Wert entgegen, welcher festlegt ob das Element dem Prozessmodell hinzugefügt oder entfernt werden soll. Diese nutzen wiederum Funktionen, die nach dem folgende Listing näher beschrieben werden.

```
1  const changeDetailLevel = (level) => {
2      setDetailLevel(level);
3      switch (level) {
4          case "0":
5              changeAnnotations(false);
6              changeDataObjects(false);
7              changeDataStores(false);
8              changeOverlay(false);
9              break;
10         case "1":
11             changeAnnotations(false);
12             changeDataObjects(true);
13             changeDataStores(true);
14             changeOverlay(false);
15             break;
16         case "2":
17             changeAnnotations(true);
18             changeDataObjects(true);
19             changeDataStores(true);
20             changeOverlay(false);
21             break;
22         default :
23             changeAnnotations(true);
24             changeDataObjects(true);
25             changeDataStores(true);
26             changeOverlay(true);
27             break;
28     }
29 };
```

Listing 6.9: Funktion zur Einstellung der Detailstufe über den Regler

Listing 6.10 zeigt exemplarisch die Implementierung der Funktion *changeAnnotations* (Zeile 1). Da Annotationen zu der Kategorie *Shapes* gehören, wird die Funktionen *changeShape* in Zeile 5 mit sämtlichen Parametern aufgerufen. Analog dazu sind Funktionen implementiert, die die Sichtbarkeit von Verbindungen und anderen Elementen regeln. Dadurch, dass die Implementierungen kaum abweichen, werden diese nicht gesondert betrachtet. Die Funktion *changeShape* überprüft in Zeile 6, ob die Annotationen bereits in dem Status sind, in den sie überführt werden sollen. Ist dies der Fall, wird die Funktion zurückgegeben ohne Änderungen vorzunehmen (Zeile 7). Falls nicht, werden die bereits vorgestellten Grundfunktionen aus Listing 6.8 angewandt (Zeile 10, 12). Für Overlays werden die Funktionen aus Listing 6.7 verwendet.

```
1  const changeAnnotations = (showIt) => {
2      changeShape(allAnnotations, showAnnotations,
           setShowAnnotations, showIt);
3  };
4
5  const changeShape = (shapeArray, showShapeArray,
       setShowShapeArray, showIt) => {
6      if (showShapeArray === showIt) {
7          return;
8      }
9      if (showIt) {
10         addElements(shapeArray);
11     } else {
12         removeElements(shapeArray);
13     }
14     setShowShapeArray(showIt);
15 };
```

Listing 6.10: Funktionen zur Einstellung der Sichtbarkeit von Annotationen

Abbildung 6.7 zeigt den Prozess zur Verabreichung eines Anästhetikums (Abbildung 2.13) auf niedrigster Detailstufe. Der Regler steht auf dem untersten Level. Links unten ist die Ebene mit blauer Farbe hinterlegt, auf der sich die Betrachtenden zu diesem Zeitpunkt befinden. Des Weiteren ist in der linken oberen Ecke über die Feinsteuerung zu sehen, welche BPMN 2.0 Elemente ein- bzw. ausgeblendet sind. In diesem beispielhaften Prozessmodell sind Annotationen sowie Datenobjekte ausgeblendet. Mit einem Klick auf das Symbol rechts unten über dem Bpmn.io Logo wird der Prozess zentriert und die Zoomstufe angepasst. Die Steuerung für die farbliche Markierung der Swimlanes ist auf der rechten oberen Seite zu sehen. Diese beinhaltet sämtliche Pools und Lanes, wobei zu diesem Zeitpunkt keines der Elemente farblich hervorgehoben ist.

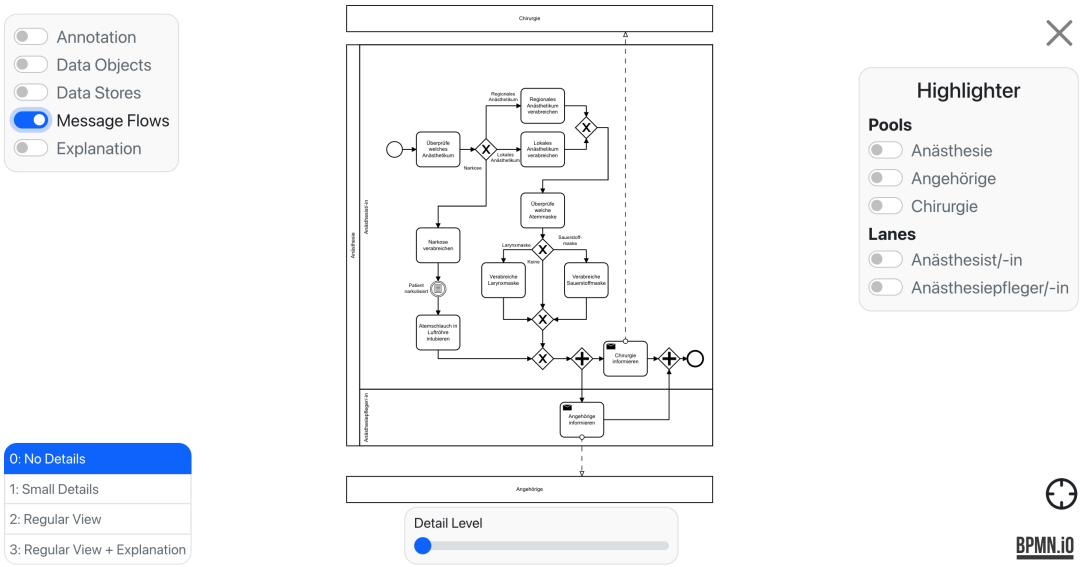

Abbildung 6.7: Betrachtung eines Prozesses auf niedrigster Detailstufe

Zur Veranschaulichung der unterschiedlichen Detailstufen, die in Listing 6.9 zu sehen sind, wird zuerst ein Prozessmodell vorgestellt. Anschließend werden die unterschiedlichen Ebenen mit dem Modell anhand von Screenshots gezeigt. Abbildung 6.8 zeigt einen Prozess für das Erstellen und Versenden einer Rechnung. Dazu wird zuerst auf die Bestellung im System zugegriffen und die Rechnung erstellt. Diese Rechnung wird dann an den Kunden versendet. Damit endet der Prozess.

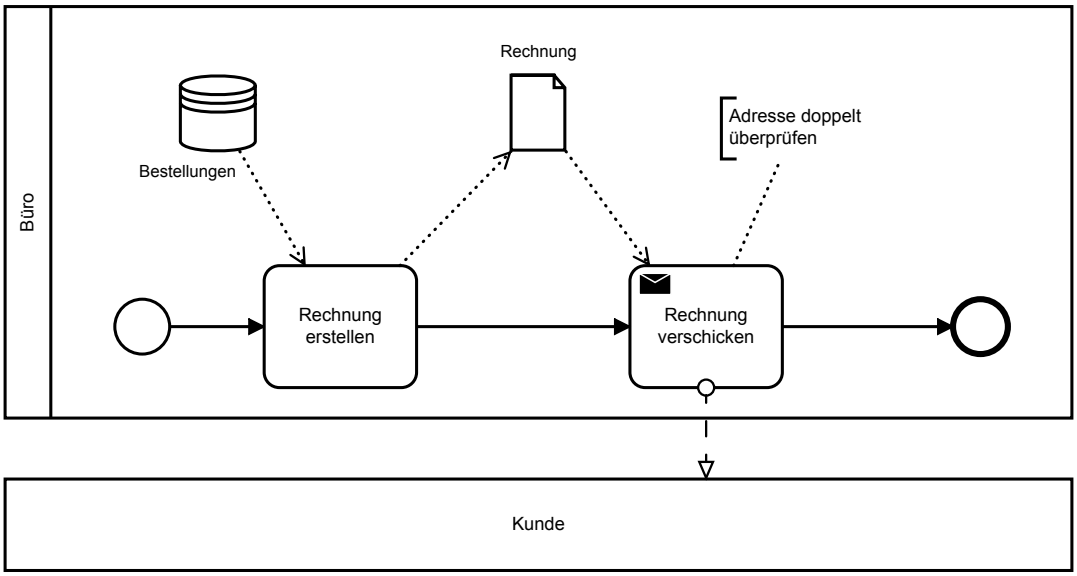

Abbildung 6.8: Prozessmodell für das Erstellen und Versenden einer Rechnung

Detailstufe 0

Abbildung 6.9 zeigt das Prozessmodell auf der niedrigsten Detailstufe. Hier sind Annotationen, Datenfluss sowie Erklärungen ausgeblendet.

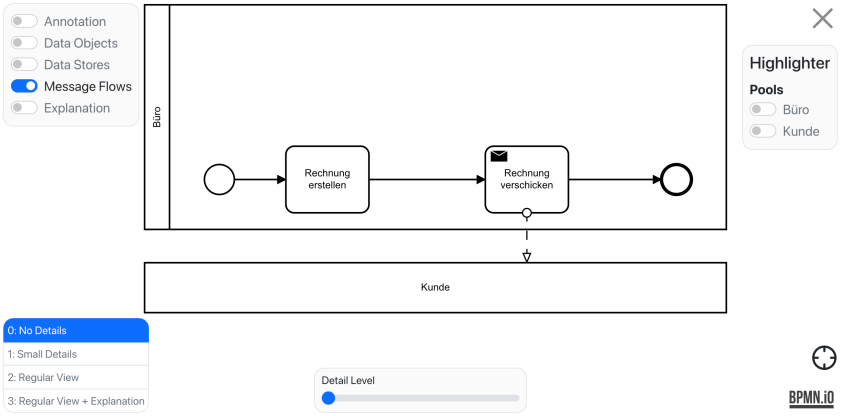

Abbildung 6.9: Betrachtung eines Prozesses auf Detailstufe 0

Detailstufe 1

Abbildung 6.10 zeigt das Prozessmodell auf der ersten Detailstufe. Hier sind Annotationen sowie Erklärungen ausgeblendet. Datenobjekte und Datenspeicher sind sichtbar.

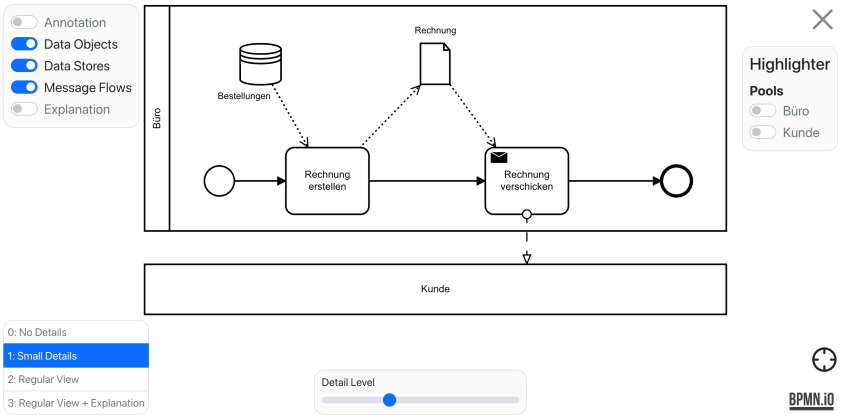

Abbildung 6.10: Betrachtung eines Prozesses auf Detailstufe 1

Detailstufe 2

In Abbildung 6.11 wird das Prozessmodell im Originalzustand dargestellt. Hier sind alle BPMN 2.0 Elemente sichtbar. Erklärungen sind ausgeblendet.

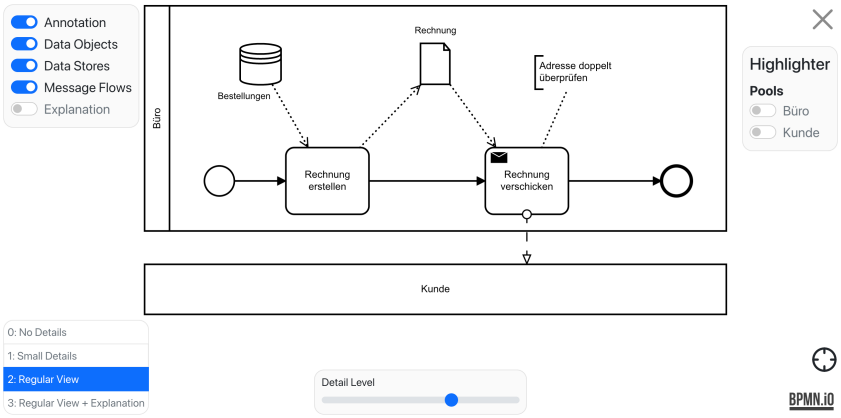

Abbildung 6.11: Betrachtung eines Prozesses auf Detailstufe 2

Detailstufe 3

Abbildung 6.12 stellt das Prozessmodell im Originalzustand dar. Hier sind alle BPMN 2.0 Elemente sichtbar. Zusätzlich dazu sind die Erklärungen eingeblendet.

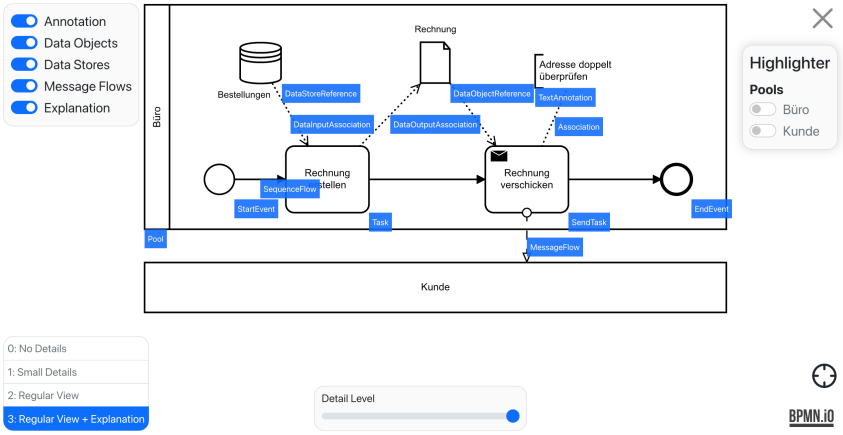

Abbildung 6.12: Betrachtung eines Prozesses auf Detailstufe 3

7 Anforderungsabgleich

In diesem Kapitel wird überprüft ob die Anforderungen an die Applikation, die in Kapitel 7 festgelegt wurden, erfüllt sind. Dazu werden zuerst die funktionalen Anforderungen betrachtet und anschließend die nicht-funktionalen Anforderungen. Der Abgleich findet dabei in Tabellenform statt.

7.1 Funktionale Anforderungen

Die folgende Tabelle zeigt den Abgleich der funktionalen Anforderungen mit der Webanwendung. Falls die Anforderung erfüllt ist, befindet sich in der rechten Spalte ein Verweis auf das jeweilige Konzept.

Tabelle 7.1: Abgleich der funktionalen Anforderungen mit Verweis auf das jeweilige Konzept

Codierung	**Anforderung**	**Konzept**
F01	Datei einlesen	Erfüllt. Siehe Datei hochladen
F02	Format überprüfen	Erfüllt. Siehe Siehe Datei hochladen und Fehlermeldung
F03	Visualisierung	Erfüllt. Siehe BPMN 2.0 Prozessmodell
F04	Einteilung Detaillierungsgrad	Erfüllt. Siehe Detail-Regler
F05	Detail-Regler	Erfüllt. Siehe Detail-Regler
F06	Sichtbarkeit	Erfüllt. Siehe Objekt Sichtbarkeit

Codierung	Anforderung	Konzept
F07	Navigieren	Erfüllt. Siehe BPMN 2.0 Prozessmodell
F08	Benennung	Erfüllt. Siehe Objekt Sichtbarkeit und Detail-Regler
F09	Detailstufen	Erfüllt. Siehe Aktuelles Level
F10	Auflistung	Erfüllt. Siehe Highlighting
F11	Hervorhebung	Erfüllt. Siehe Highlighting
F12	Ansicht zurücksetzen	Erfüllt. Siehe Ansicht zurücksetzen

7.2 Nicht-funktionale Anforderungen

Die folgende Tabelle zeigt den Abgleich der nicht-funktionalen Anforderungen mit der Webanwendung. Falls die Anforderung erfüllt ist, befindet sich in der rechten Spalte ein Verweis auf das jeweilige Konzept.

Tabelle 7.2: Abgleich der nicht-funktionalen Anforderungen mit Verweis auf das jeweilige Konzept

Codierung	Anforderung	Konzept
Usability:		
NFU01	Aufgabenangemessenheit	Erfüllt. Siehe Zusammenspiel
NFU02	Selbstbeschreibungsfähigkeit	Erfüllt. Siehe Mockup, Datei hochladen, Erklärung und Ansicht zurücksetzen
NFU03	Lernförderlichkeit	Erfüllt. Siehe Styling und Ansicht zurücksetzen
NFU04	Fehlertoleranz	Erfüllt. Siehe Ansicht zurücksetzen und Datei hochladen

Codierung	Anforderung	Konzept
NFU05	Erwartungskonformität	Erfüllt. Siehe Styling
NFU06	Individualisierbarkeit	Erfüllt. Siehe Objekt Sichtbarkeit
NFU07	Steuerbarkeit	Erfüllt. Siehe Ansicht zurücksetzen und Fehlermeldung
System:		
NFS01	Plattformübergreifend	Erfüllt. Siehe Grundgerüst
NFS02	Single-Page Applikation	Erfüllt. Siehe Grundgerüst
NFS03	Wartbarkeit	Erfüllt. Siehe React
NFS04	Skalierbarkeit	Erfüllt. Siehe Grundgerüst
NFS05	Offline	Erfüllt. Siehe Grundgerüst

8 Zusammenfassung und Ausblick

In diesem Kapitel wird die Arbeit kurz zusammengefasst. Außerdem wird im letzten Abschnitt darauf eingegangen, wie die implementierte Anwendung um weitere Funktionalitäten ergänzt und verbessert werden könnte.

8.1 Zusammenfassung

In dieser Arbeit ist eine Anwendung implementiert worden, die es ermöglicht, BPMN 2.0 Prozessmodelle zu betrachten und den Informationsgehalt dabei dynamisch zu verändern. Die entstandene Anwendung ist eine Single-Page React Webapplikation. Dadurch ist diese plattformübergreifend verwendbar und nach abgeschlossenem Ladevorgang offline nutzbar. Durch die Verwendung von React als modernes Benutzeroberflächen Framework und der Aufteilung in Komponenten ist die Anwendung geeignet für Erweiterungen und Wartungen. Außerdem wurde bei der Realisierung auf die Usability Normen Rücksicht genommen.

Um ein Prozessmodell zu betrachten, muss dieses als BPMN 2.0 valide XML-Datei verfügbar sein. Diese Datei kann dann entweder per Drag & Drop oder per Dateiauswahl hochgeladen werden. Zur Darstellung des Prozessmodells und um darin navigieren zu können, wird das Framework Bpmn.io verwendet.

Mittels realisierten Reglers ist es möglich, während der Betrachtung einzelne BPMN 2.0 Elemente wie Annotationen, Datenspeicher und Datenobjekte ein- und auszublenden. Der Sequenzfluss wird dabei nicht beeinträchtigt. Außerdem ist es zusätzlich möglich, die Sichtbarkeit dieser Elemente einzeln per Feinsteuerung zu regeln. Ergänzend dazu erlaubt die Feinsteuerung auch das Ein- und Ausblenden von Nachrichtenflüssen. Auf der höchsten Detailstufe wird das Prozessmodell im modellierten Zustand angezeigt. Zusätzlich dazu sind die Erklärungen per Overlay

sichtbar. Diese Overlays sind nahe am Element positioniert und dienen dazu, die Betrachtenden beim Verständnis des Prozessmodells zu unterstützen. Sämtliche Manipulationen des Prozessmodells müssen nach der Vorverarbeitung mit Bpmn.io interagieren. Die Interaktion findet lokal im Browser statt. Des Weiteren können mit der Anwendung einzelne Swimlanes farblich hervorgehoben werden.

Nahezu die komplette Anwendung konnte mit React Komponenten implementiert werden. Eine Ausnahme stellten die Overlays dar. Bpmn.io erwartet beim Hinzufügen von Overlays zwingend ein HTML-Element. Dadurch ist die Möglichkeit zur Realisierung von Overlays eingeschränkt und der Inhalt kann nicht dynamisch über React manipuliert werden.

8.2 Ausblick

In diesem Abschnitt werden Möglichkeiten vorgestellt, die Webanwendung um Funktionalitäten zu erweitern. Diese Ideen sind während der Bearbeitung und Implementierung dieser Abschlussarbeit aufgekommen.

8.2.1 Speicherung von Prozessmodellen

Eine denkbare Erweiterung dieser Anwendung ist eine Anbindung an eine Datenbank. In dieser Datenbank sollen die hochgeladenen Prozessmodelle abgespeichert werden. Diese sollen dann nach dem Upload über eine Benutzeroberfläche abrufbar sein. Das bietet den Vorteil, dass das Prozessmodell nicht immer wieder neu hochgeladen werden muss. Das würde sich besonders anbieten, wenn die Webanwendung im Intranet eines Unternehmens verfügbar ist und die Mitarbeiter dadurch auf sämtliche Abläufe im Unternehmen zugreifen könnten. Die Auflistung könnte dann anhand von Arbeitsbereichen, Abteilungen oder Aufgabengebieten kategorisiert werden.

8.2.2 Overlay Erklärungen

In der aktuellen Version der Implementierung haben die einzelnen BPMN 2.0 Elemente lediglich ihren Typ innerhalb des Overlay angezeigt. Um weitere Informationen über das Element herauszufinden, müssen Benutzende sich die Informationen über andere Quellen beschaffen. Jedoch wäre eine denkbare Erweiterung der Overlays, das diese auf Mausklick den Bereich vergrößern und Erklärungen zu den Elementen darstellen. Damit ist es möglich, den Einstieg zu erleichtern oder wissen aufzufrischen.

8.2.3 Dynamische Einstellungen

Die Anwendenden sollen die Möglichkeit haben, über ein gesondertes Menü Einstellungen vorzunehmen. Bestandteil dieser Einstellung könnte beispielsweise sein, von welchen BPMN 2.0 Elementen die Sichtbarkeit verstellt werden darf. Außerdem ist es denkbar, eine Standardsicht einzurichten, also damit festzulegen, welche Elemente nach dem initialen Rendern sichtbar sein sollen. Eine weitere Möglichkeit ist es, zu definieren, welche Bedienelemente bei der Betrachtung verfügbar sein müssen.

8.2.4 Ausblenden von Swimlanes

Die aktuelle Version der Webanwendung ermöglicht es, einzelne Pools und Lanes farblich zu markieren und damit hervorzuheben. Dadurch kann beispielsweise der Fokus bewusst auf eine Lane gelegt werden, weil der Betrachtende sich ausschließlich dafür interessiert. Je nach Prozessgröße können umliegende Lanes oder Pools trotzdem ablenkend wirken. Deshalb wäre eine denkbare Erweiterung, ausschließlich die Swimlane darzustellen, die angewählt ist und alle anderen komplett auszublenden.

Literatur

[1] Thomas Allweyer. *BPMN 2.0 - Introduction to the Standard for Business Process Modeling*. Norderstedt: BoD – Books on Demand, 2016. ISBN: 978-3-837-09331-5.

[2] Ralph Bobrik, Manfred Reichert und Thomas Bauer. „View-Based Process Visualization“. In: *Business Process Management*. Hrsg. von Gustavo Alonso, Peter Dadam und Michael Rosemann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 88–95. ISBN: 978-3-540-75183-0.

[3] Juliana Bowles, Ricardo M. Czekster und Thais Webber. „Annotated BPMN Models for Optimised Healthcare Resource Planning“. In: *Software Technologies: Applications and Foundations*. Hrsg. von Manuel Mazzara, Iulian Ober und Gwen Salaün. Cham: Springer International Publishing, 2018, S. 146–162. ISBN: 978-3-030-04771-9.

[4] Camunda. *Web-based tooling for BPMN, DMN and CMMN.* (o. D.) URL: https://bpmn.io/ (besucht am 08. 02. 2021).

[5] Camunda. *bpmn-js*. (o. D.) URL: https://bpmn.io/toolkit/bpmn-js/walkthrough/ (besucht am 08. 02. 2021).

[6] Issam Chebbi, Schahram Dustdar und Samir Tata. „The view-based approach to dynamic inter-organizational workflow cooperation“. In: *Data & Knowledge Engineering* 56 (Feb. 2006), S. 139–173. DOI: 10.1016/j.datak.2005.03.008.

[7] Michele Chinosi und Alberto Trombetta. „BPMN: An introduction to the standard“. In: *Computer Standards & Interfaces* 34.1 (2012), S. 124–134. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2011.06.002.

[8] Dickson K. W. Chiu u. a. „Workflow View Driven Cross-Organizational Interoperability in a Web-Service Environment“. In: *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*. CAiSE '02/ WES '02. Berlin, Heidelberg: Springer-Verlag, 2002, 41–56. ISBN: 978-3-540-00198-0.

[9] D. Choi. *Full-Stack React, TypeScript, and Node: Build Cloud-Ready Web Applications Using React 17 with Hooks and GraphQL*. Packt Publishing, Limited, 2020. ISBN: 9781839219931. URL: https://books.google.de/books?id=F90fzgEACAAJ (besucht am 12. 02. 2021).

[10] Selim Erol. „Coloring support for process diagrams: a review of color theory and a prototypical implementation“. In: *Vienna University of Economics and Business* (2015).

[11] Facebook. *Create React App*. (o. D.) URL: https://github.com/facebook/create-react-app (besucht am 19. 02. 2021).

[12] Facebook. *Create a New React App*. (o. D.) URL: https://reactjs.org/docs/create-a-new-react-app.html (besucht am 19. 02. 2021).

[13] Facebook. *Hooks at a Glance*. (o. D.) URL: https://reactjs.org/docs/hooks-overview.html (besucht am 13. 02. 2021).

[14] Facebook. *React.Component*. (o. D.) URL: https://reactjs.org/docs/react-component.html (besucht am 12. 02. 2021).

[15] H. Giraudel und M. Suzanne. *Jump Start Sass*. SitePoint Pty., 2016. ISBN: 9781457199851.

[16] D. Griffiths, D. Griffiths und an O'Reilly Media Company Safari. *React Cookbook*. O'Reilly Media, Incorporated, 2021. URL: https://books.google.de/books?id=thLgzQEACAAJ (besucht am 12. 02. 2021).

[17] B. Griggs. *Node Cookbook: Discover solutions, techniques, and best practices for server-side web development with Node.js 14, 4th Edition*. Packt Publishing, 2020. ISBN: 9781838554576. URL: https://books.google.de/books?id=kW8LEAAAQBAJ (besucht am 12. 02. 2021).

[18] Roland Groza u. a. *react-dropzone*. (o. D.) URL: https://react-dropzone.js.org/ (besucht am 27. 02. 2021).

[19] S. Hatton. „Choosing the Right Prioritisation Method“. In: *19th Australian Conference on Software Engineering (aswec 2008)*. 2008, S. 517–526. DOI: 10.1109/ASWEC.2008.4483241.

[20] D. Herron. *Node.js Web Development - Fifth Edition: Server-side Web Development Made Easy with Node 14 Using Practical Examples*. Packt Publishing, 2020. ISBN: 9781838987572. URL: https://books.google.de/books?id=7YG6zQEACAAJ (besucht am 12. 02. 2021).

[21] Markus Hipp, Bela Mutschler und Manfred Reichert. „Navigating in Complex Business Processes“. In: *Proc. 23rd Int'l Conf on Database and Expert Systems Applications (DEXA'12), Part II*. LNCS 7447. Springer, 2012, S. 466–480. URL: http://dbis.eprints.uni-ulm.de/817/ (besucht am 02. 03. 2021).

[22] Marta Indulska u. a. „Business Process Modeling: Perceived Benefits“. In: *Conceptual Modeling - ER 2009*. Hrsg. von Alberto H. F. Laender u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 458–471. ISBN: 978-3-642-04840-1.

[23] Eleanna Kafeza, Dickson K. W. Chiu und Irene Kafeza. „View-Based Contracts in an E-Service Cross-Organizational Workflow Environment“. In: *Proceedings of the Second International Workshop on Technologies for E-Services*. TES '01. Berlin, Heidelberg: Springer-Verlag, 2001, 74–88. ISBN: 354-0425-65-9.

[24] Mateja Kocbek Bule u. a. „Business Process Model and Notation: The Current State of Affairs“. In: *Computer Science and Information Systems* 12 (Juni 2015), S. 509–539. DOI: 10.2298/CSIS140610006K.

[25] Jens Kolb, Klaus Kammerer und Manfred Reichert. „Updatable Process Views for Adapting Large Process Models: The proView Demonstrator“. In: *Demo Track of the 10th Int'l Conf on Business Process Management (BPM'12)*. CEUR Workshop Proceedings 940. 2012, S. 6–11. URL: http://dbis.eprints.uni-ulm.de/847/ (besucht am 05. 02. 2021).

[26] Jens Kolb und Manfred Reichert. „A Flexible Approach for Abstracting and Personalizing Large Business Process Models“. In: *Applied Computing Review* 13.1 (2013), S. 6–17. URL: http://dbis.eprints.uni-ulm.de/914/ (besucht am 05. 02. 2021).

[27] I. Koppel. *Entwicklung einer Online-Diagnostik für die Alphabetisierung: Eine Design-Based Research-Studie*. Research (Wiesbaden, Germany). Springer Fachmedien Wiesbaden, 2016. ISBN: 9783658157692. URL: https://books.google.de/books?id=n4JJDQAAQBAJ (besucht am 13. 02. 2021).

[28] J. Krause. *Introducing Bootstrap 4: Create Powerful Web Applications Using Bootstrap 4.5*. Apress, 2020. ISBN: 9781484262023. URL: https://books.google.de/books?id=fAuWzQEACAAJ (besucht am 14. 02. 2021).

[29] Tyge-F. Kummer, Jan Recker und Jan Mendling. „Enhancing understandability of process models through cultural-dependent color adjustments“. In: *Decision Support Systems* 87 (2016), S. 1–12. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2016.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0167923616300574 (besucht am 05. 02. 2021).

[30] A. Libby. *Introducing Dart Sass: A Practical Introduction to the Replacement for Sass, Built on Dart*. Apress, 2019. ISBN: 9781484243725. URL: https://books.google.de/books?id=3JqHDwAAQBAJ (besucht am 10. 02. 2021).

[31] P. Matsinopoulos. *Practical Bootstrap: Learn to Develop Responsively with One of the Most Popular CSS Frameworks*. Apress, 2020. ISBN: 978-1-484-26070-8. URL: https://books.google.de/books?id=A46LzQEACAAJ (besucht am 14. 02. 2021).

[32] P. H. Meland und E. A. Gjære. „Representing Threats in BPMN 2.0“. In: *2012 Seventh International Conference on Availability, Reliability and Security*. 2012, S. 542–550. DOI: 10.1109/ARES.2012.13.

[33] Jan Mendling, Hajo A. Reijers und Jorge Cardoso. „What Makes Process Models Understandable?“ In: *Business Process Management*. Hrsg. von Gustavo Alonso, Peter Dadam und Michael Rosemann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 48–63. ISBN: 978-3-540-75183-0.

[34] Richard Müller und Andreas Rogge-Solti. „BPMN for Healthcare Processes“. In: *Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS 2011, Karlsruhe, Germany, February 21–22, 2011*. Hrsg. von Daniel Eichhorn, Agnes Koschmider und Huayu Zhang. Bd. 705. CEUR Workshop Proceedings. CEUR-WS.org, 2011, S. 65–72.

[35] Christine Natschläger. „Deontic BPMN“. In: *Database and Expert Systems Applications*. Hrsg. von Abdelkader Hameurlain u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 264–278. ISBN: 978-3-642-23091-2.

[36] J. Nielsen und H. Loranger. *Web Usability : Deutsche Ausgabe*. Design publishing imaging - DPI. Addison-Wesley, 2008. ISBN: 9783827327635. URL: https://books.google.de/books?id=IY_x6LpRyZAC (besucht am 13. 02. 2021).

[37] Deutsche Institut für Normung e.V. *DIN EN ISO 9241-110:2020-10 Ergonomie der Mensch-System-Interaktion - Teil 110: Interaktionsprinzipien*. Beuth Verlag, Berlin, Okt. 2020.

[38] OMG. *Business Process Model and Notation (BPMN), Version 2.0.2*. Object Management Group, 9. Dez. 2013. URL: https://www.omg.org/spec/BPMN/2.0.2/PDF (besucht am 09. 02. 2021).

[39] J. Padolsey. *Clean Code in JavaScript: Develop reliable, maintainable, and robust JavaScript*. Packt Publishing, 2020. ISBN: 9781789957297. URL: https://books.google.de/books?id=DprLDwAAQBAJ (besucht am 13. 02. 2021).

[40] E. Porcello, A. Banks und an O'Reilly Media Company Safari. *Learning React, 2nd Edition*. O'Reilly Media, Incorporated, 2019. URL: https://books.google.de/books?id=BBBOzQEACAAJ (besucht am 11. 02. 2021).

[41] S.F. Rahman u. a. *Your First Week with Bootstrap*. SitePoint Pty., 2018. ISBN: 9781925836035. URL: https://books.google.de/books?id=vNwAvgEACAAJ (besucht am 14. 02. 2021).

[42] Jan Recker u. a. „Business Process Modeling- A Comparative Analysis“. In: *Journal of the Association of Information Systems* 10 (Apr. 2009). DOI: 10.17705/1jais.00193.

[43] H. A. Reijers und J. Mendling. „A Study Into the Factors That Influence the Understandability of Business Process Models“. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 41.3 (2011), S. 449–462. DOI: 10.1109/TSMCA.2010.2087017.

[44] S.B. Rinderle u. a. „Business Process Visualization - Use Cases, Challenges, Solutions“. In: *Proceedings of the Eighth International Conference on Enterprise Information Systems (ICEIS'06): Information System Analysis and Specification*. Hrsg. von Y. Manolopoulos u. a. INSTICC PRESS, Mai 2006, S. 204–211. ISBN: 972-8865-43-0.

[45] W. Schneider. *Ergonomische Gestaltung von Benutzungsschnittstellen: Kommentar zur Grundsatznorm DIN EN ISO 9241-110*. Beuth Kommentar. Beuth Verlag GmbH, 2008. ISBN: 9783410164951. URL: https://books.google.de/books?id=wVaOsiBaUtwC (besucht am 13. 02. 2021).

[46] Wolfgang Schneider. *Grundsätze der Dialoggestaltung nach DIN EN ISO 9241-110*. 1. Sep. 2018. URL: https://www.ergo-online.de/ergonomie-und-gesundheit/software/dialoggestaltung/grundsaetze-der-dialoggestaltung-nach-din-en-iso-9241-110/grundsaetze-der-dialoggestaltung-nach-din-en-iso-9241-110/ (besucht am 12. 02. 2021).

[47] Matthias Schrepfer u. a. „The Impact of Secondary Notation on Process Model Understanding“. In: *The Practice of Enterprise Modeling*. Hrsg. von Anne Persson und Janis Stirna. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 161–175. ISBN: 978-3-642-05352-8.

[48] Karsten A. Schulz und Maria E. Orlowska. „Facilitating cross-organisational workflows with a workflow view approach“. In: *Data & Knowledge Engineering* 51.1 (2004). Contact-driven coordination and collaboration in the Internet context, S. 109–147. ISSN: 0169-023X. DOI: https://doi.org/10.1016/j.datak.2004.03.008. URL: https://www.sciencedirect.com/science/article/pii/S0169023X04000357 (besucht am 15. 02. 2021).

[49] Signavio. *Paralleles Gateway*. (o. D.) URL: https://documentation.signavio.com/suite/de/Content/workflow-accelerator/userguide/control-flow/parallel-gateway.htm (besucht am 14. 02. 2021).

[50] Vinicius Stein Dani, Carla Maria Dal Sasso Freitas und Lucinéia Heloisa Thom. „Ten years of visualization of business process models: A systematic literature review“. In: *Computer Standards & Interfaces* 66 (2019), S. 103347. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2019.04.006.

[51] C.Y. Subhash. *Introduction To Client Sever Computing*. New Age International (p) Limited, 2009. ISBN: 9788122426892. URL: https://books.google.de/books?id=AD0TjdCRVEkC (besucht am 19. 02. 2021).

[52] E.R. Tufte und Graphics Press. *Envisioning Information*. Graphics Press, 1990. ISBN: 9781930824140. URL: https://books.google.de/books?id=fW9jAAAAMAAJ (besucht am 02. 03. 2021).

[53] S.A. White und D. Miers. *BPMN Modeling and Reference Guide: Understanding and Using BPMN*. Business Process Management Process Modeling. Future Strategies Incorporated, 2008. ISBN: 9780977752720. URL: https://books.google.de/books?id=0Z2Td3bCYW8C (besucht am 09. 02. 2021).

Name: Florian Gallik Matrikelnummer: 788744

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 11.03.2021 .....

Florian Gallik