

Thesis

# Nachvollziehbarkeit von Nutzerinteraktion und Anwendungsverhalten am Beispiel JavaScript-basierter Webanwendungen

An der Fachhochschule Dortmund  
im Fachbereich Informatik  
Studiengang Software- und Systemtechnik  
Vertiefung Softwaretechnik  
erstellte Thesis  
zur Erlangung des akademischen Grades  
Bachelor of Science

von  
Marvin Kienitz  
geb. am 26.04.1996  
Matr.-Nr. 7097533

Betreuer:  
Prof. Dr. Sven Jörges  
Dipl.-Inf. Stephan Müller

Dortmund, 15. Februar 2021

## Kurzfassung

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.*

*Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.*

*Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.*

*Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.*

## Abstract

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.*

*Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.*

*Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.*

*Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.*

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Zielsetzung . . . . .	2
1.2.1	Abgrenzung . . . . .	2
1.3	Vorgehensweise . . . . .	3
1.4	Open Knowledge GmbH . . . . .	3
<b>2</b>	<b>Ausgangssituation</b>	<b>4</b>
2.1	Browserumgebung . . . . .	4
2.1.1	Browserprodukte . . . . .	4
2.1.2	JavaScript . . . . .	5
2.2	Single-Page-Applications . . . . .	6
2.3	Nachvollziehbarkeit . . . . .	7
2.3.1	Nachvollziehbarkeit bei SPAs . . . . .	7
2.3.2	Browserbedingte Hürden . . . . .	7
2.3.2.1	Cross-Origin Resource Sharing (CORS) . . . . .	7
2.3.2.2	Content-Security-Policy . . . . .	8
2.3.3	Logdaten . . . . .	9
2.3.4	Fernzugriff . . . . .	9
<b>3</b>	<b>Methoden und Praktiken</b>	<b>10</b>
3.1	Methoden für eine bessere Nachvollziehbarkeit . . . . .	10
3.1.1	Logging . . . . .	10
3.1.2	Metriken . . . . .	11
3.1.3	Tracing . . . . .	11
3.1.4	Fehlerberichte . . . . .	12
3.2	Praktiken . . . . .	12
3.2.1	System Monitoring . . . . .	13
3.2.2	Log Management . . . . .	13
3.2.3	Application Performance Monitoring (APM) . . . . .	13
3.2.4	Real User Monitoring (RUM) . . . . .	14
3.2.5	Error Monitoring . . . . .	14

3.2.6	Session-Replay . . . . .	14
3.2.7	OpenTelemetry . . . . .	15
3.3	Werkzeuge und Technologien . . . . .	17
3.3.1	Kriterien . . . . .	17
3.3.1.1	Funktionalitäten . . . . .	17
3.3.1.2	Open Source . . . . .	17
3.3.1.3	Standardisierung . . . . .	17
3.3.1.4	Flexibilität . . . . .	18
3.3.2	Vorauswahl . . . . .	18
3.3.3	New Relic . . . . .	18
3.3.4	Dynatrace . . . . .	19
3.3.5	Sentry . . . . .	19
3.3.6	LogRocket . . . . .	20
3.3.7	Splunk . . . . .	21
3.3.8	Honeycomb . . . . .	22
3.3.9	Jaeger . . . . .	22
3.3.10	TraVista . . . . .	23
3.3.11	FAME . . . . .	24
3.3.12	The Kaiju Project . . . . .	24
<b>4</b>	<b>Erstellung Proof-of-Concept</b>	<b>26</b>
4.1	Anforderungen . . . . .	26
4.1.1	Definitionen . . . . .	26
4.1.2	Anforderungsanalyse . . . . .	27
4.1.3	Anforderungsliste . . . . .	28
4.1.3.1	Grundanforderungen . . . . .	28
4.1.3.2	Funktionsumfang . . . . .	29
4.1.3.3	Eigenschaften . . . . .	30
4.1.3.4	Partnersysteme . . . . .	31
4.2	Vorstellung der Demoanwendung . . . . .	32
4.2.1	Verhaltensdefinition . . . . .	33
4.2.2	Backend . . . . .	37
4.2.3	Frontend . . . . .	39
4.2.3.1	Warenkorb . . . . .	39
4.2.3.2	Rechnungsadresse . . . . .	40
4.2.3.3	Lieferdaten . . . . .	41
4.2.3.4	Zahlungsdaten . . . . .	42
4.2.3.5	Bestellübersicht . . . . .	43
4.2.3.6	Bestellbestätigung . . . . .	44
4.2.4	Fehlerszenarien . . . . .	44
4.2.4.1	„Keine Übersetzungen“ . . . . .	45
4.2.4.2	„Gültige Straßen sind ungültig“ . . . . .	45

## Inhaltsverzeichnis

---

4.2.4.3	„Gültige Städte sind ungültig“ . . . . .	45
4.2.4.4	„Ungültige Adressen sind gültig“ . . . . .	45
4.2.4.5	„Vor- und Nachnamen werden abgeschnitten“ . . . . .	46
4.2.4.6	„Falsche Zahlungsart“ . . . . .	46
4.2.4.7	„Lange Verarbeitung“ . . . . .	46
4.3	Konzept . . . . .	47
4.3.1	Datenverarbeitung . . . . .	47
4.3.1.1	Erhebung . . . . .	47
4.3.1.2	Auswertung . . . . .	47
4.3.1.3	Präsentation . . . . .	48
4.3.2	Architektur . . . . .	49
4.3.3	Technologie-Stack . . . . .	50
4.3.4	Übertragbarkeit . . . . .	51
4.4	Implementierung . . . . .	52
4.4.1	Backend . . . . .	52
4.4.2	Frontend . . . . .	53
4.4.2.1	Traces und Metriken . . . . .	53
4.4.2.2	Logging . . . . .	56
4.4.2.3	Fehler . . . . .	58
4.4.2.4	Session-Replay . . . . .	59
<b>5</b>	<b>Ergebnis</b>	<b>61</b>
5.1	Demonstration . . . . .	61
5.2	Kriterien . . . . .	61
5.3	Übertragbarkeit . . . . .	61
5.4	Einschätzung von anderen Entwicklern (optional) . . . . .	61
<b>6</b>	<b>Abschluss</b>	<b>63</b>
6.1	Fazit . . . . .	63
6.2	Ausblick . . . . .	63
	<b>Anhang</b>	<b>64</b>
<b>7</b>	<b>Anhang</b>	<b>64</b>
7.1	Studien zur Browserkompatibilität . . . . .	64
	<b>Eidesstattliche Erklärung</b>	<b>65</b>
	<b>Abkürzungsverzeichnis</b>	<b>66</b>
	<b>Abbildungsverzeichnis</b>	<b>67</b>
	<b>Tabellenverzeichnis</b>	<b>68</b>

## *Inhaltsverzeichnis*

---

<b>Quellcodeverzeichnis</b>	<b>68</b>
<b>Literaturverzeichnis</b>	<b>69</b>

# 1 Einleitung

## 1.1 Motivation

Die Open Knowledge GmbH ist als branchenneutraler Softwaredienstleister in einer Vielzahl von Branchen wie Automotive, Logistik, Telekommunikation und Versicherungs- und Finanzwirtschaft aktiv. Zu den zahlreichen Kunden der Open Knowledge GmbH gehört auch ein führender deutscher Direktversicherer.

Ein Direktversicherer bietet Versicherungsprodukte seinen Kunden ausschließlich im Direktvertrieb, d. h. vor allem über das Internet und zusätzlich auch über Telefon, Fax oder Brief an. Im Unterschied zum klassischen Versicherer verfügt ein Direktversicherer jedoch über keinen Außendienst oder Geschäftsstellen, bei denen Kunden eine persönliche Beratung erhalten. Da das Internet der primäre Vertriebskanal ist, ist heute ein umfassender Online-Auftritt die Norm. Dieser besteht typischerweise aus einem Kundenportal mit der Möglichkeit Angebote für Versicherungsprodukte berechnen und abschließen zu können, sowie persönliche Daten und Verträge einsehen und ändern zu können.

Während in der Vergangenheit Online-Auftritte i. d. R. als Webanwendung mit serverseitigen Rendering realisiert wurden, sind heutzutage Javascript-basierte Webanwendungen mit clientseitigem Rendering die Norm [WC14]. Bei einer solchen Webanwendung befindet sich ein Großteil der Logik im Browser des Nutzers, bspw. wird der Nutzer durch einen komplexen Wizard geführt und erst bei Absenden des letzten Formulars geschieht eine Interaktion mit einem Server.

Im produktiven Einsatz kommt es auch bei gut getesteten Webanwendungen hin und wieder vor, dass es zu unvorhergesehenen Fehlern in der Berechnung oder Verarbeitung kommt. Liegt die Ursache für den Fehler im Browser, z. B. aufgrund einer ungültigen Wertkombination, ist dies eine Herausforderung. Während bei Server-Anwendungen Fehlermeldungen in den Log-Dateien einzusehen sind, gibt es für den Betreiber der Anwendung i. d. R. keine Möglichkeit die notwendigen Informationen über den Nutzer und seine Umgebung abzurufen. Noch wichtiger ist, dass er mitbekommt, wenn ein Nutzer ein Problem bei der Bedienung der Anwendung hat. Ohne eine aktive Benachrichtigung durch den Nutzer, sowie detaillierte Informationen, ist es dem Betreiber nicht möglich, Kenntnis über das Problem zu erlangen, geschweige denn dieses nachzustellen.



Dies stellt ein Kernproblem von Webanwendungen dar [Fil20]. Im Rahmen der Arbeit soll daher ein Proof-of-Concept konzipiert und umgesetzt werden, welcher dieses Kernproblem am Beispiel einer Demoanwendung löst.

### 1.2 Zielsetzung

Das grundlegende Ziel dieser Arbeit soll es sein, den Betreibern einer JavaScript-basierten Webanwendung die Möglichkeit zu geben, das Verhalten ihrer Anwendung und die Interaktionen von Nutzer nachzuvollziehen. Diese Nachvollziehbarkeit soll insbesondere bei Fehlerfällen u. Ä. gewährleistet sein, ist aber auch in sonstigen Fällen zu erstreben, wie z. B. wenn die Betreiber nachvollziehen wollen, welche Interaktionen der Nutzer getätigt hat. Eine vollständige Überwachung der Anwendung und des Nutzers (wie bspw. bei Werbe-Tracking) sind jedoch nicht vorgesehen. Daraus ergibt sich die Forschungsfrage:

Wie sieht ein Ansatz aus, um den Betreibern von clientseitigen JavaScript-basierten Webanwendungen eine Nachvollziehbarkeit zu gewährleisten?

Vom Leser wird eine Grundkenntnis der Informatik in Theorie oder Praxis erwartet, aber es sollen keine detaillierten Erfahrungen in der Webentwicklung vom Leser erwartet werden. Daher sind das Projektumfeld und seine besonderen Eigenschaften zu erläutern.

Die anzustrebende Lösung soll ein Proof-of-Concept sein, welches eine, zu erstellen, Demoanwendung erweitert. Die Demoanwendung soll repräsentativ eine abgespeckte JavaScript-basierte Webanwendung darstellen, bei der die zuvor benannten Hürden zur Nachvollziehbarkeit bestehen.

Weiterhin gilt es zu beleuchten, wie die Auswirkungen für die Nutzer der Webanwendung sind. Wurde die Leistung der Webanwendung beeinträchtigt (erhöhte Ladezeit, erhöhte Datenlast)? Werden mehr Daten von ihm erhoben und zu welchem Zweck?

Am Ende der Ausarbeitung soll überprüft werden, ob und wie die Forschungsfrage beantwortet wurde. Auch die Übertragbarkeit der erstellten Lösung (PoC) und Ergebnisse gilt es hierbei näher zu betrachten.

#### 1.2.1 Abgrenzung

Die Demoanwendung wird als Single-Page-Application (SPA) [JSD15] realisiert, denn hier bewegt sich das Projektumfeld von der Open Knowledge GmbH. Bei der Datenerhebung und -verarbeitung sind datenschutzrechtliche Aspekte nicht näher zu betrachten. Bei der Betrachtung von Technologien aus der Wirtschaft ist eine bewertende Gegenüberstellung nicht das Ziel.

### 1.3 Vorgehensweise

Um das Ziel dieser Arbeit, also ein Proof-of-Concept zu erstellen, welches die Nachvollziehbarkeit einer bestehenden Anwendung erhöht, zu erreichen, wird zunächst die theoretische Seite des Forschungsfeldes beleuchtet. Hierzu gehört eine nähere Betrachtung der Umgebung „Browser“ und von Webanwendungen, sowie gilt es die Nachvollziehbarkeit zu definieren und im Hinblick auf SPAs zu erläutern.

Darauf aufbauend sind aktuelle Ansätze zur verbesserten Nachvollziehbarkeit zu recherchieren und zu beschreiben. Speziell sollen hierbei die allgemeinen übergreifenden Methoden und die tatsächlichen angewandten Praktiken differenziert beschrieben werden. Hierbei ist u. A. der Stand der Technik aus Wirtschaft und Literatur zu erläutern, um darauffolgend und auf Basis dessen ein Proof-of-Concept zu erstellen.

Bevor jedoch der PoC implementiert wird, soll ein Konzept erstellt werden, welches darlegt, wie der PoC eine verbesserte Nachvollziehbarkeit erreicht. Ist nun das Konzept erstellt, gilt es dieses auf eine SPA anzuwenden und das Proof-of-Concept zu erstellen. Im Anschluss an die Implementierung gilt es diese kritisch zu bewerten, einerseits ob die Forschungsfrage beantwortet werden konnte und andererseits in Aspekten wie Übertragbarkeit und Auswirkungen für den Nutzer.

### 1.4 Open Knowledge GmbH

Die Bachelorarbeit wird im Rahmen einer Werkstudententätigkeit innerhalb der Open Knowledge GmbH erstellt. Der Standortleiter des Standortes Essen, Dipl.-Inf. Stephan Müller, übernimmt die Zweitbetreuung.

Die Open Knowledge GmbH ist ein branchenneutrales mittelständisches Dienstleistungsunternehmen mit dem Ziel bei der Analyse, Planung und Durchführung von Softwareprojekten zu unterstützen. Das Unternehmen wurde im Jahr 2000 in Oldenburg, dem Hauptsitz des Unternehmens, gegründet und beschäftigt heute 74 Mitarbeiter. Mitte 2017 wurde in Essen der zweite Standort eröffnet, an dem 13 Mitarbeiter angestellt sind.

Die Mitarbeiter von Open Knowledge übernehmen in Kundenprojekten Aufgaben bei der Analyse über die Projektziele und der aktuellen Ausgangssituationen, der Konzeption der geplanten Software, sowie der anschließenden Implementierung. Die erstellten Softwarelösungen stellen Individuallösungen dar und werden den Bedürfnissen der einzelnen Kunden entsprechend konzipiert und implementiert. Technisch liegt die Spezialisierung bei der Mobile- und bei der Java Enterprise Entwicklung, bei der stets moderne Technologien und Konzepte verwendet werden. Die Geschäftsführer als auch diverse Mitarbeiter der Open Knowledge GmbH sind als Redner auf Fachmessen wie der Javaland oder als Autoren in Fachzeitschriften wie dem Java Magazin vertreten.

## 2 Ausgangssituation

### 2.1 Browserumgebung

Web Browser haben sich seit der Veröffentlichung von Mosaic, einer der ersten populären Browser, im Jahr 1993 stark weiterentwickelt [Kop14]. Das Abrufen und Anzeigen von statischen HTML-Dokumenten wurde mithilfe von JavaScript um interaktive und später um dynamische Inhalte erweitert. Heutzutage können Entwickler komplexe Webanwendungen realisieren [JSD15], welche zudem browserunabhängig entwickelt werden können. Durch diese Entwicklung und die vielseitigen Anwendungsfälle, besitzt die Umgebung „Browser“ besondere Eigenschaften, welche nachfolgend beschrieben werden.

#### 2.1.1 Browserprodukte

Die Vielfalt an Browsern bereitet Webentwicklern immer wieder eine Herausforderung, nämlich ob ein von ihnen bereitgestelltes Produkt für die Nutzer einwandfrei funktioniert, unabhängig der Browserpräferenz des Nutzers. Die Häufigkeit solcher Probleme, auch Cross-Browser-Incompatibilities (XBI) [CPO14] genannt, hat jedoch abgenommen. Dies ist unter anderem durch den Trend von offenen Web-Standards, wie die des W3C [W3C20], erklärbar.

Generell lässt sich feststellen, dass auch in der Literatur die Veröffentlichungen in Bezug auf (In-)Kompatibilität von Browsern abnehmen, wie in Abbildung 2.1 zu betrachten. Dies spricht dafür, dass das Problem von XBIs weniger präsent ist als zuvor. Somit wird die besondere Hürde, die XBIs darstellen, nicht als eine relevante Hürde in dieser Arbeit angesehen.

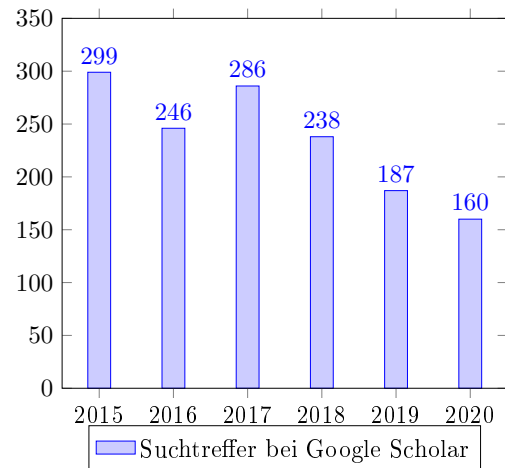


Abb. 2.1: Studien zur Browserkompatibilität, eigene Darstellung (vgl. 7.1)

Im Jahr 2020 gab es weitere Entwicklungen, die die Kompatibilität zwischen Browsern erhöhte. Microsoft ist beim Folgeprodukt zum Internet Explorer, dem Microsoft Edge Browser, von einer proprietären Browser-Engine zu Chromium gewechselt [Mic20b] und verwendet denselben Kern wie Chrome und Opera. Ende 2020 wurde zudem der Support für den Internet Explorer 11 eingestellt [Mic20a]. Im Januar 2021 meldete StatCounter [Sta21] eine Marktverteilung bei Desktop-Browsern von 65,96% Chrome, 10,43% Safari, 8,39% Firefox, 7,43% Edge, 2,59% Opera und 2,54% Internet Explorer.

### 2.1.2 JavaScript

Als JavaScript 1997 veröffentlicht und in den NetScape Navigator integriert wurde, gab es die Bedenken, dass das Öffnen einer Webseite dem Betreiber erlaubt Code auf dem System eines Nutzers auszuführen. Damit dies nicht eintritt, wurde der JavaScript Ausführungskontext in eine virtuelle Umgebung integriert, einer sog. Sandbox [Pow06].

Die JavaScript-Sandbox bei Browsern schränkt u. A. den Zugriff auf das Dateisystem ein. Auch Zugriff auf native Bibliotheken oder Ausführung von nativem Code ist nicht möglich [OKSK15]. Browser bieten darüber hinaus aber einige Schnittstellen an, die es erlauben z. B. Daten beim Client zu speichern oder auch Videos abzuspielen.

1999 nahm Microsoft im Internet Explorer 5.0 eine neue Methode in ihre JavaScript-Umgebung auf: Ajax (Asynchronous JavaScript and XML) [MM21]. Ajax erlaubt die Datenabfrage von Webservern mittels JavaScript. Hierdurch können Inhalte auf Webseiten dynamisch abgefragt und dargestellt werden, wofür zuvor ein weiterer Seitenaufruf notwendig war. Das Konzept wurde kurz darauf von allen damals gängigen Browsern übernommen. Jedoch fand erst mit der Standardisierung 2006 durch das W3C [W3C06] die Methode bei Entwicklern Anklang und ist seitdem der Grundstein für unser dynamisches und interaktives Web [Hop06].

Durch dies wurden Webanwendungen immer beliebter, aber Entwickler klagten darüber, dass Browser die Abfragen von JavaScript nur auf dem bereitstellenden Webserver, also „same-origin“, erlauben [Ran09]. Um dies zu ermöglichen, wurde im selben Jahr der Standardisierung von Ajax ein erster Entwurf zur Absicherung von Abrufen domänenfremder Ressourcen eingereicht [OPBW06], das sogenannte Cross-Origin Resource Sharing.

Über die Jahre wurde der JavaScript-Standard immer umfangreicher, was Entwicklern erlaubte mächtige Werkzeuge sowie Frameworks zu entwickeln, welche die Erstellung von Webanwendungen vereinfachen. Mit Webanwendungen war es nun möglich, einen großen Teil der Funktionalitäten eines Produktes abzubilden. Diese „clientbasierten“ Anwendungen werden im nächsten Abschnitt näher beleuchtet.

## 2.2 Single-Page-Applications

Single-Page-Applications (SPAs) sind eine Teilmenge von JavaScript-basierten Webanwendungen (JS WebApps). Bei letzterem handelt es sich um Webanwendungen, die in JavaScript realisiert wurden und welche den Browser als Laufzeitumgebung verwenden [Kie10]. Die mit JavaScript realisierten Funktionen umfassen u. A. Interaktivität und dynamische Inhaltsdarstellung. Hierbei werden meist nur Grundgerüste der Anwendung in HTML und CSS bereitgestellt, jedoch werden die eigentlichen Inhalte dynamisch mit JavaScript erstellt. Die Inhalte werden auch über zusätzliche Schnittstellen der Webanwendung zu Partnersystemen bereitgestellt.

SPAs gehen bei der dynamischen Inhaltsdarstellung jedoch einen Schritt weiter als JS WebApps [JSD15]. Die gesamte Anwendung wird über ein einziges HTML-Dokument und die darin referenzierten Inhalte erzeugt. Oftmals wird auf Basis dessen ein erheblicher Teil der Logik auf Clientseite umgesetzt, was die Anwendung zu einem Rich- bzw. Fat-Client macht.

Für das Bereitstellen einer solchen Anwendung ist meist nur ein simpler Webserver ausreichend und ein oder mehrere Dienste, von denen aus die SPA ihre Inhalte abrufen kann. Populäre Frameworks zur Realisierung von SPAs sind beispielsweise Angular [Goo20], React [Fac20b] oder Vue.js [YM20].

SPAs bieten zudem durch ihre clientbasierte Herangehensweise Stakeholdern die Möglichkeit, die Anwendung als Offline-Version bereitzustellen. Sind neben der Logik keine externen Daten notwendig oder wurden diese bereits abgerufen und gecached, so kann eine SPA auch „offline“ von Benutzern verwendet werden. Weiterhin steigern SPAs die User Experience (UX), indem sie u. A. schneller agieren, da keine kompletten Seitenaufrufe notwendig sind [AMWR20].

Durch diesen grundsätzlich anderen Ansatz gibt es aber auch negative Eigenschaften. Unter anderem werden native Browserfunktionen umgangen, wie die automatisch befüllte Browserhistorie, denn es werden keine neuen HTML-Dokumente angefragt. Weiterhin leiden „virtuelle“ Verlinkungen und Buttons darunter, dass sie nicht alle Funktionen unterstützen, die normale HTML-Elemente aufweisen. Um dies und andere verwandte Probleme zu beheben, besitzen die zuvor genannten Frameworks spezielle Implementierungen oder ggf. muss eine zusätzliche Bibliothek herangezogen werden, wie z. B. die jeweiligen Router-Bibliotheken.

Nichtsdestotrotz ist ein jahrelanger Trend von der Einführung von Single-Page-Applications zu erkennen, heutzutage steht eine große Auswahl an erprobten Technologien zur Verfügung [GB20].

## 2.3 Nachvollziehbarkeit

Neben der Umgebung Browser beschäftigt sich die Arbeit hauptsächlich mit der Nachvollziehbarkeit. Nachvollziehbarkeit bedeutet allgemein, dass über ein resultierendes Verhalten eines Systems auch interne Zustände nachvollzogen werden können. Dies ist keine neue Idee, sondern fand bereits 1960 im Gebiet der Kontrolltheorie starke Bedeutung [Ká60]. Nach Freedman [Fre91] und Scrocca *et al.* [STM<sup>+</sup>20] lässt sich diese Definition auch auf Softwaresysteme übertragen und wird dabei mit „Observability“ bezeichnet. Scrocca adaptiert dabei die von Majors [Maj18] genannte Definition:

„Observability for software is the property of knowing what is happening inside a distributed application at runtime by simply asking questions from the outside and without the need to modify its code to gain insights.“

Insbesondere in der Wirtschaft hat sich der Begriff der Observability etabliert [Fra20] [Rel21]. Hierbei lässt sich die Observability als eine Weiterentwicklung des klassischen Monitoring von Software betrachten [Wat18]. Ziel dabei ist es Anwendungen und Systeme weitestgehend beobachtbar zu machen und darauf basierend Betreibern und Entwicklern zu ermöglichen, auch aus unbekannten Situationen Rückschlüsse über die Anwendung oder das System ziehen zu können [?].

### 2.3.1 Nachvollziehbarkeit bei SPAs

Speziell in dieser Arbeit wird die Nachvollziehbarkeit bei Webanwendungen näher betrachtet. Wie zuvor in Abschnitt 2.2 geschildert, gibt es bei Webanwendungen und insbesondere Single-Page-Applications besondere Eigenschaften, die es den Betreibern und Entwicklern erschweren das Verhalten ihrer Anwendung und die Interaktionen eines Nutzers nachzuvollziehen. Meist lassen sich aus Sicht der Betreiber nur die Kommunikationsaufrufe der Anwendung zum Backend nachvollziehen, aber nicht wie es dazu gekommen ist und wie diese Daten weiterverarbeitet werden. Somit ist eine gängige SPA nicht gut nachvollziehbar.

### 2.3.2 Browserbedingte Hürden

#### 2.3.2.1 Cross-Origin Resource Sharing (CORS)

Wie aus der Geschichte zu JavaScript zu sehen ist, entwickelte CORS sich aus dem Wunsch von Entwicklern, nicht auf einen einzelnen Webserver beschränkt zu sein. Diese Einschränkung existierte, um Nutzer vor Missbrauch zu schützen. CORS hebt diese Einschränkung teilweise auf, aber unter Berücksichtigung der sicherheitskritischen Aspekte.

Das Konzept von CORS stellt sicher, dass aus einer JavaScript-Umgebung heraus keine Ressourcen von Webservern angefragt werden, welche nicht explizit der Anfrage zustimmen [MM20c].

Wie eine „cross-origin“ Ajax-Anfrage nach dem Konzept von CORS gehandhabt wird, ist in Abbildung 2.2 zu betrachten. Wenn eine HTTP-Anfrage nicht „simple“<sup>1</sup> ist, führt der Browser einen sogenannten „Preflighted Request“ aus, bei dem vor der eigentlichen Anfrage eine zusätzliche OPTIONS-Anfrage gesendet wird. Bestätigt nun der Webserver in seiner Antwort auf die OPTIONS-Anfrage, dass die Anfrage so erlaubt ist, wird auch die eigentliche Ajax-Anfrage ausgeführt. Ansonsten schlägt die Anfrage fehl und im JavaScript-Kontext ist lediglich der Fehlschlag zu sehen, ohne einen Hinweis auf die Diskrepanz bzgl. CORS.

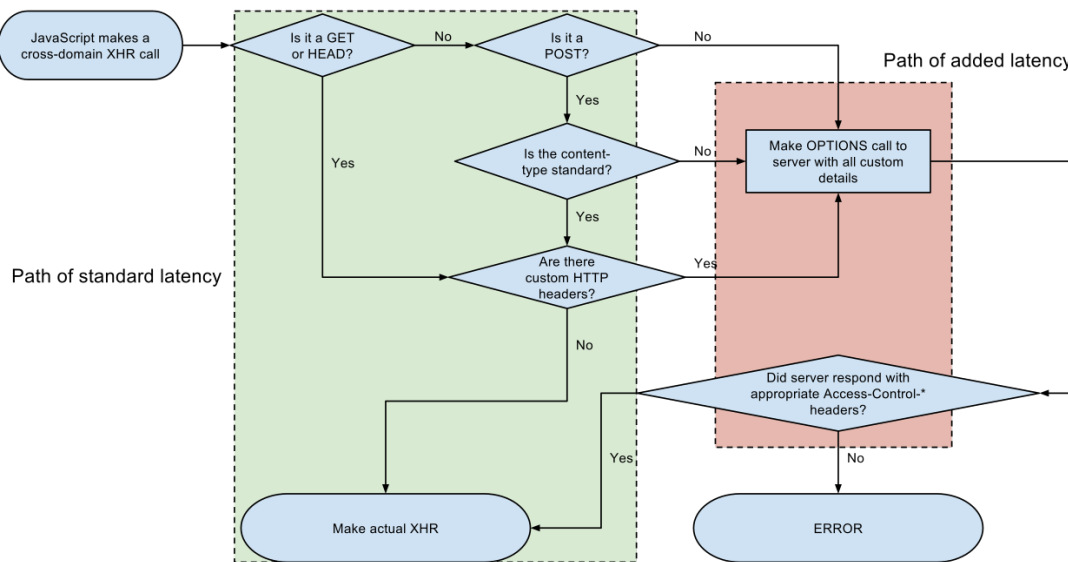


Abb. 2.2: Flowchart über den Ablauf von Ajax-Anfragen mit CORS [Blu15]

### 2.3.2.2 Content-Security-Policy

Neben CORS gibt es im Browser eine Möglichkeit zu bestimmen, welche Funktionalitäten einer Webanwendung zur Verfügung stehen und wie diese vom Browser einzuschränken sind. Diese Funktion heißt Content-Security-Policy (CSP) und dient unter anderem dem Schutz vor Cross-Site-Scripting, indem eine Webanwendung beschränken kann, welche

<sup>1</sup>Eine Anfrage ist „simple“, wenn 1. der Methode GET, HEAD oder POST entspricht; 2. sie keine eigene Header enthält; und 3. der „Content-Type“ von POST-Anfragen einem der folgenden Werte entspricht: „application/x-www-form-urlencoded“, „multipart/form-data“ oder „text/plain“ [MM20c].

Funktionalitäten in JavaScript verfügbar sind und von wo aus Skripte und Daten geladen werden dürfen [MM20b]. Weiterhin kann bei einem Versuch diese Regeln zu umgehen, eine Berichterstattung darüber eingerichtet werden.

### 2.3.3 Logdaten

Ähnlich wie bei anderen Umgebungen gibt es eine standardisierte Log- bzw. Konsolenausgabe für die JavaScript-Umgebung [MM20a]. Diese Ausgabe ist aber für den Standard-Benutzer unbekannt und es kann nicht erwartet werden, dass Nutzer dieses Log bereitstellen. Durch die zuvor beschriebenen Härtungsmaßnahmen von Browsern ist es hinzukommend nicht möglich, das Log direkt in eine Datei zu schreiben.

Um die Logdaten also zu erheben, gilt es entweder ein spezielles Log-Framework in der Webanwendung zu verwenden oder die bestehende Schnittstelle zu überschreiben oder zu wrappen. Nachdem die Datenerhebung gewährleistet ist, gilt es jedoch zudem die Daten an ein Partnersystem weiterzuleiten, welches die Beachtung der zuvor beschriebenen Einschränkungen erfordert. Alles in Allem stellt sich die Logdatenerhebung als nicht trivial dar, eine genauere Betrachtung erfolgt in der Untersuchung bestehender Lösungen.

### 2.3.4 Fernzugriff

Ein weiterer Punkt, der den „Browser“ von anderen Umgebungen unterscheidet, ist, dass die Betreiber und Entwickler sich normalerweise nicht auf die Systeme der Nutzer schalten können. Bei Expertenanwendungen, bei denen die Nutzerschaft bekannt ist, ließe sich solch eine Funktionalität ggf. realisieren. Es gibt jedoch keine standardmäßige Funktionalität auf die gesetzt werden kann, wie z. B. das Remote Application Debugging [Ora20] von Java. Weiterhin sind bei einer Webanwendung, die für den offenen Markt geschaffen ist, hierbei sind die Nutzer zahlreich sowie unbekannt und so eine Funktionalität lässt sich nicht realistisch umsetzen.



## 3 Methoden und Praktiken

### 3.1 Methoden für eine bessere Nachvollziehbarkeit

#### 3.1.1 Logging

Mit Logging bezeichnet man die systematische Protokollierung von Softwareprozessen und ihren internen Zuständen [ZHF<sup>+</sup>15]. Diese erstellten Protokolle nennt man Logs und sie helfen Betreibern und Entwicklern nach der Ausführung einer Anwendung nachvollziehen zu können, wie die genaue Verarbeitung war. Die daraus resultierende Nachvollziehbarkeit setzt jedoch voraus, dass genügend und informationsreiche Logmeldungen in die Anwendung eingebaut wurden [ZHF<sup>+</sup>15].

Logs stellen meist die hauptsächliche oder einzige Methode dar, wie Betreiber und Entwickler das Verhalten einer Anwendung im Produktivmodus nachvollziehen können [ZHF<sup>+</sup>15]. Gerade in Problemfällen können Logs kritische Informationen bereitstellen. Bei JavaScript-basierten Webanwendungen werden jedoch selten Logs aus einer Produktivumgebung erhoben. Dies ist u. A. durch die Notwendigkeit, die Logs von einem Endnutzersystem an ein Partnersystem weiterleiten zu müssen, zu begründen, wie in Unterabschnitt 2.3.3 erläutert.

Logmeldungen erfolgen meist textbasiert und in einem menschenlesbaren Format. Wenn ein Aggregator nun jedoch Informationen aus einer großen Menge von Logs extrahiert, ist so ein Format hinderlich, da es nicht effizient analysiert werden kann. Um dem entgegen zu wirken, kommt Structured Logging ins Spiel. Bei Structured Logging [TVNP13] werden die Logmeldungen in einem vordefinierten Format erzeugt. Dieses Format kann entweder auch menschenlesbar sein oder definiert die Logmeldung bspw. als JSON-Objekt. Durch die feste Definition des Formates wird der Loganalyse ermöglicht, effizient die notwendigen Daten zu extrahieren.

Wird Structured Logging eingesetzt und ein System analysiert die Protokolldaten auf enthaltene Werte, so wird ermöglicht, dass auf diese Protokolle nicht nur manuell einzusehen sind, sondern dass auch auf Basis dessen komplexe Datenanalysen durchgeführt werden können [TVNP13]. Mit diesen Datenanalysen lassen sich auch bei großen Datenmengen situationsrelevante Informationen entlocken [LGB19]. Weiterhin lassen sich so aus Logmeldungen auch spezielle Daten wie Metriken extrahieren.

### 3.1.2 Metriken

Mit Metriken versucht man Softwareeigenschaften vergleichbar abzubilden. Hierfür werden einzelne Messungen durchgeführt, die dann mit anderen Messungen derselben Kategorie gegenübergestellt werden. Mit den Gegenüberstellungen lassen sich wiederum Rückschlüsse zu Softwareeigenschaften ziehen, dass bspw. eine Anfrage deutlich länger benötigt als andere „gleichwertige“ Anfragen. Weiterhin lassen sich historische Veränderungen in den Metriken erkennen und können unerwünschte Abweichungen aufdecken.

Beispiele für Metriken sind:

1. Eine Zeitmessung, bzgl. der Dauer einer HTTP-Anfrage.
2. Eine Messung der CPU-Auslastung.

### 3.1.3 Tracing

Tracing beschäftigt sich mit dem Aufzeichnen von Kommunikationsflüssen [OHH<sup>+</sup>16]. Hierbei erfasst Tracing einerseits die Kommunikationsflüsse innerhalb einer Anwendung bzw. innerhalb eines Systems. Andererseits zeichnet Tracing aber auch die Kommunikationsflüsse bei verteilten Systemen auf, um diese, meist komplexen Zusammenhänge, zu veranschaulichen. Ein herstellerunabhängiger Standard, der sich aus diesem Gebiet entwickelt hat, ist OpenTracing [Ope20f].

OpenTracing bildet diese Kommunikationsflüsse über zwei grundlegende Objekte ab: Traces und Spans. Ein Span besitzt einen Anfangs- und einen Endzeitpunkt und *umspannt* meist eine Methode, bei einer Webanwendung kann dies eine Verarbeitung sein oder einen durch den Nutzer hervorgerufenen Eventfluss sein. Ein Span kann Kindspans beinhalten, wenn in der Methode weitere Spans erzeugt wurden (z. B. durch einen Methodenaufruf). Ein Trace ist eine Menge von Spans, die alle über eine einzelne logische Aktion - wie z. B. den Druck einer Taste - ausgelöst wurden oder resultieren. Ein Trace lässt sich einerseits über die kausalen Beziehungen zwischen den Spans visualisieren (vgl. Abbildung 3.1), oder auch über die zeitliche Reihenfolge der einzelnen Spans (vgl. Abbildung 3.2).

Ein verteilter Trace, oftmals „Distributed Trace“ genannt, ist ein Trace, der sich aus den Spans von verschiedenen Systemen zusammensetzt, die miteinander kommunizieren. Hierbei werden die Traceinformationen oftmals über hinzugefügte Felder bei existierenden Aufrufen propagiert, wie z. B. dem Einfügen eines Trace-Headers. Die dann an ein Tracesystem gemeldeten Spans gehen somit über die Grenzen von Anwendungen, Prozessen und Netzwerken hinaus und bilden somit einen „Distributed Trace“ [Ope20e]. Auf Basis von reellen Aufrufen können somit die tatsächlichen Zusammenhänge der einzelnen Systeme miteinander nachempfunden werden.

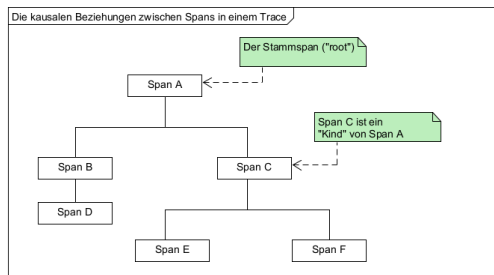


Abb. 3.1: Kausale Beziehung zwischen Spans. Eigene Darstellung.

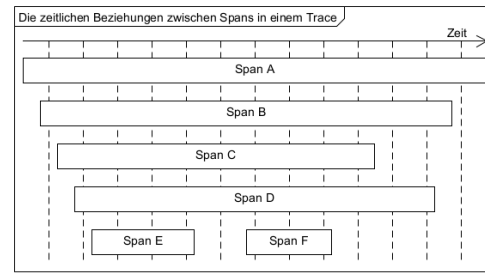


Abb. 3.2: Zeitliche Beziehung zwischen Spans. Eigene Darstellung.

#### 3.1.4 Fehlerberichte

Fehlerberichte sind ein klassisches Mittel, um den Nutzer selbst aktiv werden zu lassen und zu erfragen, welche Aktionen er durchgeführt hat und was schiefgelaufen ist (vgl. Abbildung 3.3). Hiermit können Fehler, aber auch unverständliche Workflows, aufgedeckt werden. Weiterhin kann die Intention des Nutzers ermittelt werden, vorausgesetzt er gibt dies an.

Konträr zu diesen Vorteilen stehen jedoch die von Bettenburg *et al.* [BJS<sup>+</sup>08] gefunden Ergebnisse über die Effektivität von Fehlerberichten. Denn Nutzer meldeten Informationen und Details, die sich für die Entwickler als nicht allzu hilfreich herausstellten. Diese Diskrepanz kann u. A. dadurch erläutert werden, dass Nutzer im Regelfall kein technisches Verständnis des Systems vorweisen.

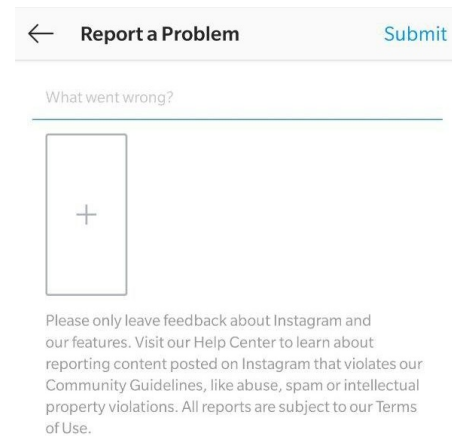


Abb. 3.3: Fehlerbericht in der Instagram App [Fac20a]

## 3.2 Praktiken

In der Fachpraxis haben sich einige Technologien über die Jahre entwickelt und etabliert, welche die Nachvollziehbarkeit von Anwendungsverhalten und Nutzerinteraktion ermöglichen oder verbessern. Auf Basis der zuvor vorgestellten Methoden und teils neuer Ansätze haben sich in der Wirtschaft einige Praktiken entwickelt. Diese werden folgend näher beleuchtet.

### 3.2.1 System Monitoring

System Monitoring beschäftigt sich mit der Überwachung der notwendigen Systeme und Dienste in Bezug auf Hardware- und Softwareressourcen. Es handelt sich hierbei um ein projektunabhängiges Monitoring, welches sicherstellen soll, dass die Infrastruktur funktionstüchtig bleibt.

Ein Beispiel für System Monitoring wäre u. A., wenn man eine Menge an Systemen auf die Festplatten- und CPU-Auslastung hin kontrolliert und überwacht. Es können weitere Aspekte überwacht werden, aber im Regelfall hat die Überwachung selbst nichts mit einem eigentlichen Projekt zu tun, außer dass die Infrastruktur hierfür sichergestellt wird.

### 3.2.2 Log Management

Log Management umfasst die Erfassung, Speicherung, Verarbeitung und Analyse von Logdaten von Anwendungen. Neben diesen Funktionen bieten solche Werkzeuge oftmals fundierte Suchfunktionen und Visualisierungsmöglichkeiten. Um die Daten aus einer Anwendung heraus zu exportieren, gibt es meist eine Vielzahl an Integrationen für Frameworks und Logbibliotheken.

Einer der wichtigsten Punkte beim Log Management, ist der Umgang mit großen Datenmengen und die gewünschten Operationen, die Nutzer damit durchführen möchten.

### 3.2.3 Application Performance Monitoring (APM)

Beim Application Performance Monitoring werden Messdaten innerhalb von Anwendungen gesammelt, um das Anwendungsverhalten nachvollziehbar zu gestalten [ABC<sup>+</sup>16]. Beispielsweise werden Aufrufe von Schnittstellen näher beleuchtet und die Antwortzeit gespeichert. Auf Basis der Daten lassen sich u. A. Abweichungen von der Norm feststellen, von einzelnen Systemen oder vom aktuellen Gesamtsystem zu einem vorherigen Zeitpunkt.

Mithilfe von APM lassen sich allgemeine Aspekte von Software, wie die Ressourcennutzung, überprüfen aber auch spezielle Faktoren, wie die Ausführungszeit einer wichtigen Methode, lassen sich so beleuchten. Die zu veranschaulichenden Aspekte werden zum Großteil über Metriken in numerische Werte abgebildet.

### 3.2.4 Real User Monitoring (RUM)

Real User Monitoring beschäftigt sich mit dem Mitschneiden von allen Nutzerinteraktionen und Umgebungseigenschaften einer Benutzeroberfläche [CGL<sup>+</sup>15]. Hiermit lässt sich nachvollziehen, wie ein Nutzer die Anwendung verwendet. RUM kann weiterhin dazu verwendet werden, um nachzuvollziehen, wie ein Zustand vom Nutzer erreicht worden ist. Aber es können auch ineffiziente Klickpfade hierdurch festgestellt werden und darauf basierend UX-Verbesserungen vorgenommen werden.

Weiterhin ist es beim RUM auch üblich Nutzerinteraktionen gruppierte zu analysieren, um verbreitete und unübliche Sitzungen zu identifizieren. Hiermit wird die Nutzerschaft und ihre Verhaltensweisen nachvollziehbarer gemacht.

### 3.2.5 Error Monitoring

Das Error Monitoring konzentriert sich auf das Erfassen und Melden von Fehlern [BT19]. Neben den eigentlichen Fehlern werden meist alle verfügbaren Kontextinformationen mit erfasst. Darunter finden sich u. A. Daten aus den Gebieten RUM und Logging. Das Error Monitoring wird oftmals eng mit einem Issue-Management verbunden, um aufgetretene Fehler und deren Behebung nachvollziehbar zu machen [BT19].

### 3.2.6 Session-Replay

Session-Replay beschreibt das Vorgehen, eine Sitzung eines Nutzers nachzustellen, so als ob sie gerade passiert [EAN17]. Hierbei können einzelne Aspekte der Anwendung nachgestellt werden, bspw. der Kommunikationsablauf oder die DOM-Manipulationen. Je mehr Aspekte nachgestellt werden, desto realitätsnaher ist die Nachstellung und entsprechend hilfreich ist sie beim Nachvollziehen.

Realitätsnahes Session-Replay nimmt somit eine enorme Datenmenge für jede Nutzersitzung auf und benötigt besonders bei Browsern eine effiziente Kommunikation, um die UX nicht negativ zu beeinflussen.

Bereits 2013 entwickelten Burg *et al.* [BBKE13] mit „Timelapse“ ein Framework, um Benutzersitzungen bei Webanwendungen aufzunehmen und wiederzugeben. Timelapse unterscheidet sich zu gängigen Session-Replay-Ansätzen dahingehend, dass die Wiedergabe keine vereinfachte Nachstellung der Anwendung ist. Stattdessen wird die JavaScript-Eventloop abgekapselt und es werden die Aufrufe

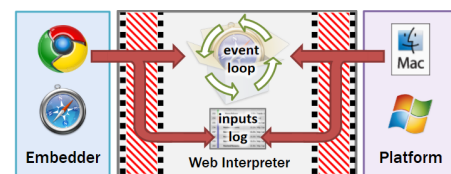


Abb. 3.4: Mitschneiden von DOM-Events, aus [BBKE13]

von und zu der Eventloop mitgeschnitten (vgl. Abbildung 3.4).

Beim Abspielen werden die Aufrufe dann in derselben Reihenfolge an die Eventloop übergeben (vgl. Abbildung 3.5). Dies bedeutet es ist ein exaktes wiederholtes Ausführen in derselben Umgebung möglich und dies ermöglicht eine detaillierte Nachvollziehbarkeit des Anwendungsverhaltens. Leider wird für diesen Ansatz eine gepatchte Version von WebKit vorausgesetzt, somit wird auch Zugriff auf das Endnutzersystem benötigt. Aus diesem Grund und weil es sehr mehr als 5 Jahren nicht mehr gepflegt wird<sup>1</sup>, ist es ungeeignet für die hier angestrebte Lösung. Die vorgestellten Konzepte stellen jedoch nützliche Kernprinzipien für das Session Replay im Allgemeinen dar.

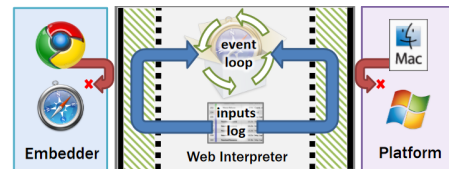


Abb. 3.5: Abspielen von DOM-Events, Abb. aus [BBKE13]

#### 3.2.7 OpenTelemetry

OpenTelemetry (OTel) [Ope20b] ist ein sich derzeit<sup>2</sup> entwickelnder herstellerunabhängiger Standard, um Tracing-, Metrik- und Logdaten<sup>3</sup> zu erfassen, zu verarbeiten, zu analysieren und zu visualisieren. OTel fasst die beiden Standards OpenTracing und OpenCensus [Ope20a] zusammen und hat sich als Ziel gesetzt diese zu erweitern [Jos19]. Hinter dem Standard stehen u. A. die Cloud Native Computing Foundation (CNCF), Google, Microsoft, und führende Hersteller von Tracing- und Monitoring-Lösungen.

Ziel ist es, dass Entwickler Tools und Werkzeuge benutzen können, ohne erneut hochspezifische Anbindungen schreiben und konfigurieren zu müssen. Stattdessen definiert der Standard Komponenten, die spezielle Aufgabengebiete haben und mit einer allgemeinen API anzusprechen sind. Die technische Infrastruktur einer auf OTel basierenden Lösung ist in Abbildung 3.6 zu sehen. Im groben definiert OTel folgende Komponenten: API, SDK, Exporter, Collector und Backend (vgl. Abbildung 3.7).

<sup>1</sup>Timelapse GitHub Repo <https://github.com/bug/replay-staging/>

<sup>2</sup>Ein erster (General-Availability-)Release der Spezifikation ist für Q1 2021 geplant [Ope21b].

<sup>3</sup>Eine Entwicklung einer Definition zu Logging ist im Gange [Ope21c].

## OpenTelemetry Collection

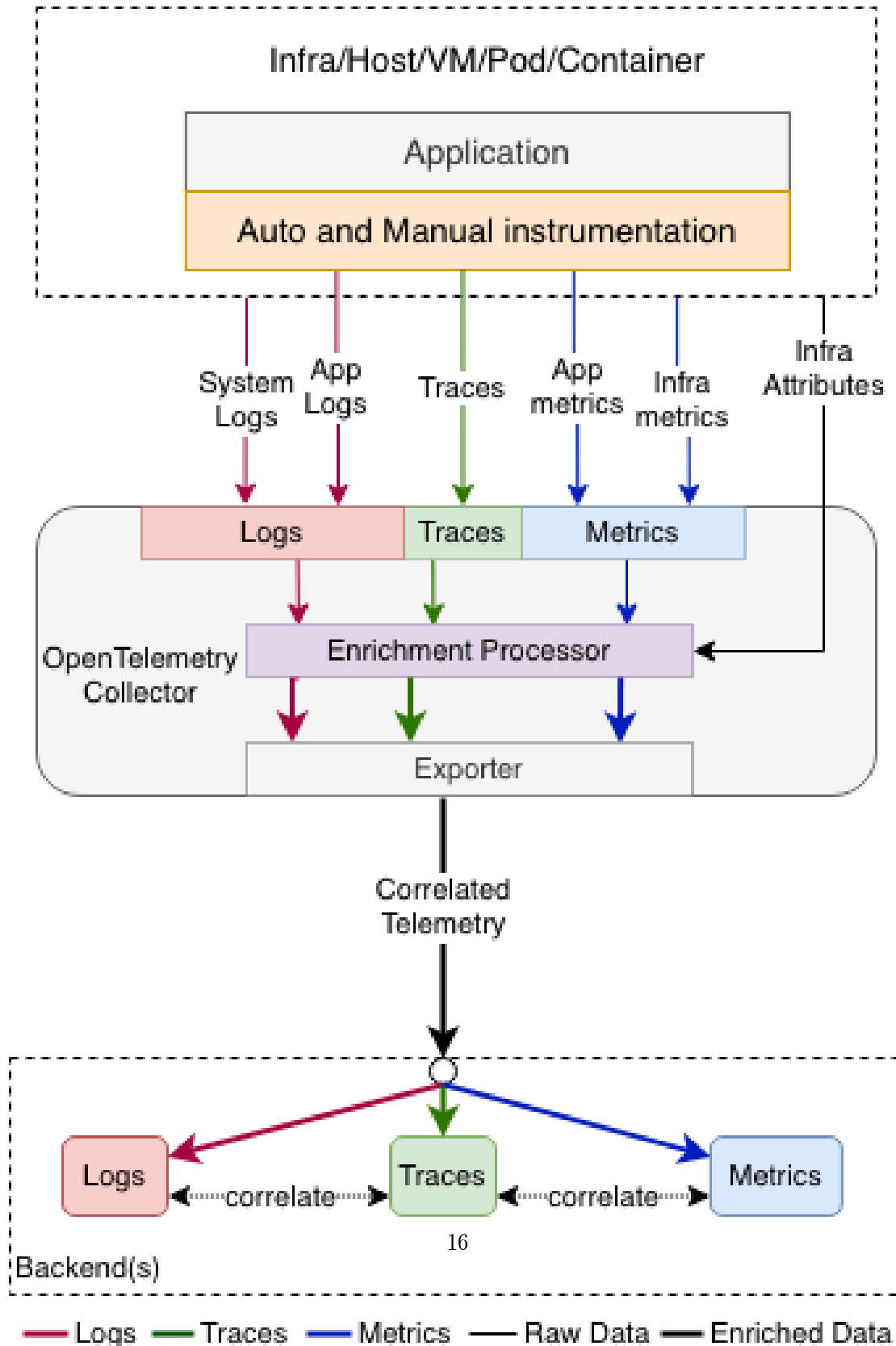


Abb. 3.6: Schaubild einer Lösung auf Basis von OTEL [Ope20c]

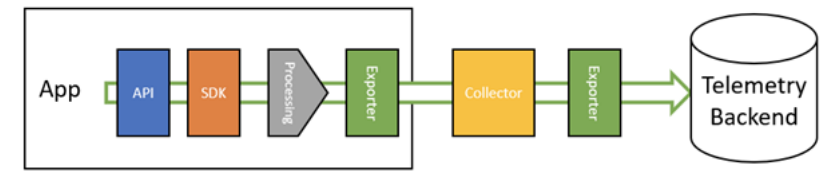


Abb. 3.7: OTEL Komponenten [Dyn20c]

## 3.3 Werkzeuge und Technologien

### 3.3.1 Kriterien

Um in dem Proof-of-Concept auf die hier vorgestellten Technologien zurückgreifen zu können, werden diese auf Basis verschiedener Kriterien bewertet. Diese Kriterien reichen von objektiven Eigenschaften bis hin zu subjektiven Einschätzungen, die bei der jeweiligen Evaluierung entstanden sind. Die Kriterien werden folgend näher beschrieben.

#### 3.3.1.1 Funktionalitäten

Das Kriterium „Funktionalitäten“ befasst sich damit, welche Probleme diese Technologie lösen kann. Hierbei ist zu bewerten, welche der folgenden Funktionalitäten die Technologie besitzt: Tracing, Erhebung von Fehlerberichten von Nutzern, System Monitoring, Log Management, APM, RUM, Error Monitoring oder Session-Replay.

#### 3.3.1.2 Open Source

Zunächst soll bewertet werden, ob und wie die einzelnen Komponenten der jeweiligen Technologie quelloffen veröffentlicht wurden. Dabei ist zu beleuchten, ob mit außenstehenden Entwicklern diese Komponenten weiterentwickelt werden oder ob die quelloffene Veröffentlichung lediglich einen Spiegelung der internen Entwicklung darstellt.

#### 3.3.1.3 Standardisierung

Bei dem Kriterium „Standardisierung“ wird bewertet, wie sehr die Technologie auf Standards wie OpenTelemetry o. Ä. setzt. Beispielsweise ist zu bewerten, ob die Daten auf eine Weise erhoben werden, die solchen Standards entsprechen oder ob auch Komponenten ausgetauscht werden können.



### 3.3.1.4 Flexibilität

Unterschiedliche betriebliche Bedingungen erfordern meist eine hohe „Flexibilität“ von einer neuen Technologie, damit diese ohne hohen Aufwand in ein bestehendes System integriert werden kann. Hierbei ist unter anderem zu bewerten, wie ein Unternehmen die Technologie aufsetzen kann - ist eine „OnPremise“-Lösung möglich, also ein Aufsetzen in der eigenen Infrastruktur, oder existiert sie nur als Software-as-a-Service (SaaS). Weiterhin ist zu bewerten, ob die Technologie auch anpassungsfähig ist auf Änderungswünsche, wenn bspw. neue Sichten zu den bestehenden Daten gefordert werden.

### 3.3.2 Vorauswahl

In dieser Arbeit werden nicht alle auf dem Markt verfügbaren Technologien näher betrachtet und bewertet. Hauptgrund hierfür ist der hohe Zeitaufwand, den eine Evaluierung mit tatsächlichem Einsatz der Technologie mit sich bringt.

Die Betrachtung einer Technologie ist somit nicht als Empfehlung dessen zu verstehen. Die nicht näher betrachteten, aber als dennoch interessant empfundenen Technologien, werden folgend aufgezählt:

- APM & RUM: AppDynamics, DataDog
- Error Monitoring: Airbrake, Instabug, Rollbar, Bugsnag, TrackJS
- Tracing oder Observability: Google Cloud Trace, Zipkin, Logz.io, Lightstep

### 3.3.3 New Relic

New Relic [New20b] ist eine SaaS-Lösung der gleichnamigen Firma, welcher Betreiber von Softwareprojekten dabei unterstützt das Verhalten ihrer Anwendungen zu überwachen. Der Dienst konzentriert sich auf APM. Neben den Kernfunktionalitäten unterstützt New Relic auch System Monitoring, RUM, Log Management, Tracing und Error Monitoring.

New Relic steht sowohl als kostenlose sowie als kommerzielle Lösung zur Verfügung. Da die kostenlose Version benötigte Features abdeckt, wurde diese im Rahmen der Arbeit eingesetzt. Der New Relic Agent, welcher die Daten beim Client sammelt, wird über ein Skript eingebunden und sendet in regelmäßigen Abständen Daten an New Relic. Über die Oberfläche von New Relic können dann allgemeine Charakteristika der Clients betrachtet werden, wie Ladezeiten, Browserhersteller, Ajax-Antwortzeiten. Spezielle Eigenschaften eines einzelnen Nutzers sind jedoch nicht möglich auszulesen. Im Fehlerfall stehen aber mehr Informationen zur Verfügung, wie der Stacktrace, die genaue Browserversion und die Uhrzeit, welche die Oberfläche veranschaulicht.

Mithilfe dieser Informationen erhält ein Entwickler oder Betreiber einen guten Rahmen, um die Umgebung als solches zu verstehen und den eigentlichen Fehler einzusehen. Ein Manko ist aber, dass diese Informationen nicht Aufschluss darüber geben, wie es zu dem Fehler kam, also welche Ereignisse direkt dem Fehler vorgelagert waren - es sind unter anderem keine Logdaten einzusehen.

New Relic bietet darüber hinaus eine Unterstützung von OTel-Daten [New20a], welches jedoch nicht mit evaluiert werden konnte, da es in der kostenlosen Version nicht enthalten war. Auf der offiziellen Seite von OTel werden hingegen zumindest öffentliche Exporter für New Relic angeboten [Ope20d].

Funktionalitäten APM, System Monitoring, RUM, Log Management, Tracing und Error Monitoring	Open Source teilweise	Standardisierung teilweise	Flexibilität e
---	--------------------------	-------------------------------	-------------------

#### 3.3.4 Dynatrace

Dynatrace [Dyn20b] ist eine SaaS-Lösung des gleichnamigen Unternehmens und ähnelt im Funktionsumfang stark New Relic. Es werden wie bei New Relic die Kernfunktionalitäten APM und RUM, sowie die Disziplinen Log Management, Tracing und Error Monitoring angeboten.

Dynatrace bietet neben dem kostenpflichtigen Dienst eine 14-tägige Testversion dessen, welche im Rahmen dieser Arbeit verwendet wurde. Die Datenerhebung erfolgt über den Dynatrace OneAgent, welcher genauso wie New Relics Agent kontinuierlich Daten sendet. In der Oberfläche von Dynatrace sind die gleichen Datenkategorien zu finden, wie bei New Relic. Dynatrace bietet zudem auch die Funktionalität des Error Monitoring, aber leidet unter demselben Problem wie New Relic: zu wenig Kontextinformationen.

Dynatrace ist zudem dem OpenTelemetry Team beigetreten und hat angegeben, an der Weiterentwicklung mitzuhelfen [Dyn20a]. Eine Integration des Dienstes Dynatrace ins Ökosystem von OTel gibt es jedoch noch nicht.

#### 3.3.5 Sentry

Sentry [Fun20] ist ein SaaS-Produkt der Functional Software Inc., welches sich auf das Error Monitoring spezialisiert. Die Kernfunktionalitäten beschränken sich auf das Error Monitoring, auch wenn von anderen Praktiken einige Aspekte präsent sind, stellen diese keine eigens abgeschlossene Funktionalität dar.

Neben einer kommerziellen Version, bietet Sentry auch eine unbegrenzte kostenlose Version bereit, welche im Rahmen dieser Arbeit evaluiert wurde. Um von Webanwendungen Fehler zu erfassen und an Sentry zu melden, bietet Sentry bei NPM quelloffene Pakete an [Sen20a]. Dabei werden Pakete für folgende Frameworks bereitgestellt: JavaScript, Angular, AngularJS, Backbone, Ember, Gatsby, React und Vue.

Anders als bei den beiden vorherigen Tools wird zu Sentry nur kommuniziert, wenn ein Fehler auftritt. Hierbei werden dafür aber umso mehr Daten erhoben: Detaillierte Klickpfade des Nutzers, Logmeldungen der Browserkonsole sowie die Informationen, die auch die anderen Tools bereitstellen.

In der Oberfläche von Sentry werden die gemeldeten Fehler in sogenannten „Issues“ zusammengefasst. Diese entsprechen Fehlertickets und die Oberfläche erlaubt eine Zuweisung dieser Tickets sowie ein detailliertes Nachhalten der Fehlerbehebung.

Die angebotenen Fehlerinformationen von Sentry sind zahlreich und helfen beim Nachvollziehen besser als die vorher beleuchteten Werkzeuge, jedoch mangelt es an einer ganzheitlichen Nachvollziehbarkeit, d. h. wenn kein Fehlerfall eingetreten ist, bietet Sentry hierfür auch keine Informationen.

Der Quellcode für das Backend von Sentry wurde veröffentlicht und weiterhin bietet Sentry auch eine OnPremise-Lösung, die auf Docker basiert [Sen20b].

#### 3.3.6 LogRocket

LogRocket [Log20] ist ein SaaS-Produkt des gleichnamigen Unternehmens und konzentriert sich auf detailliertes Session Replay von JavaScript-basierten Clientanwendungen, um Probleme identifizieren, nachvollziehen und lösen zu können.

LogRocket bietet eine kostenlose Testversion des SaaS-Produktes an, welche für die Evaluierung verwendet wurde. Zur Datenerhebung wurde das Paket `logrocket` von NPM hinzugezogen, welches nach der Initialisierung eigenständig die notwendigen Daten sammelt. Mithilfe dieser Daten wird die gesamte Sitzung des Nutzers nachgestellt. Hierbei ist die Anwendung, die Nutzerinteraktionen, die Netzwerkaufrufe sowie das DOM zu sehen. Die Reproduktion wird videoähnlich aufbereitet und erlaubt ein präzises Nachvollziehen der zeitlichen Reihenfolge und Bedeutung (vgl. Abbildung 3.8).

Neben dem JavaScript-SDK bietet LogRocket quelloffenene Plugins für folgende Bibliotheken: Redux, React, MobX, Vuex, ngrx, React Native. LogRocket ist zudem als OnPremise-Lösung verfügbar. Zusätzlich bietet LogRocket auch eine Integration für andere Tools, wie z. B. Sentry. Bei der Sentry-Integration wird bei einem gemeldeten Fehler direkt auf das „Video“ in LogRocket verlinkt, sodass der Fehler genau betrachtet werden kann.

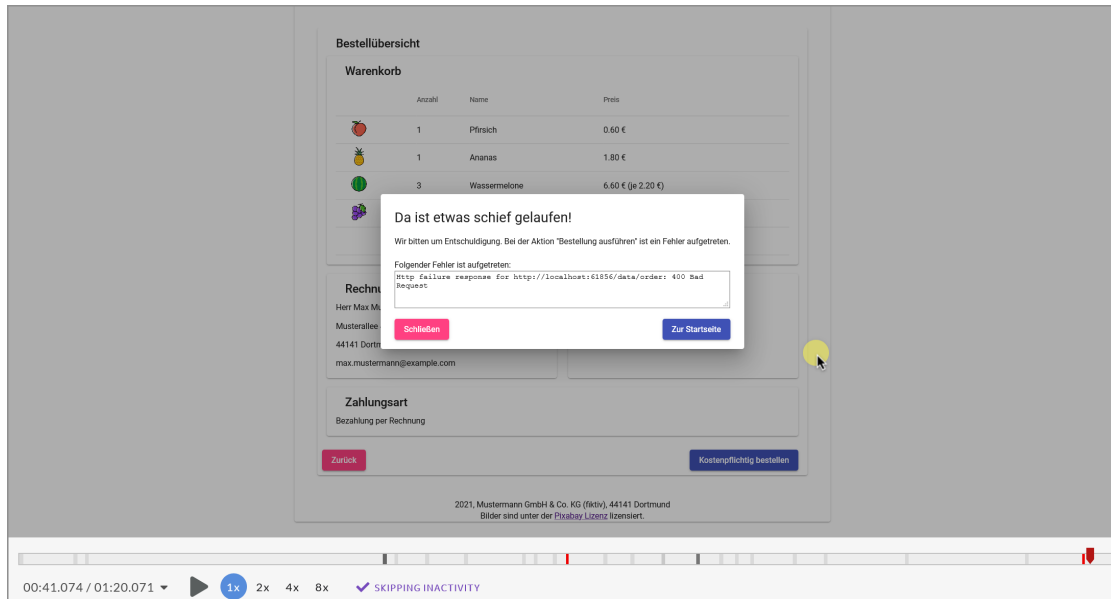


Abb. 3.8: Ausschnitt eines Session Replays bei LogRocket

#### 3.3.7 Splunk

Splunk [Spl20] ist ein Softwareprodukt und eine SaaS-Lösung des gleichnamigen Unternehmens. Splunk fungiert als eine universelle Datensinke, z. B. für Metriken und Logdaten. Es bietet Funktionen diese Daten zu durchsuchen, überwachen, analysieren und visualisieren. Weiterhin wird Splunk klassisch für Log Management eingesetzt, kann aber auch Aspekte von APM, RUM und Error Monitoring erfüllen.

Splunk bietet neben kostenpflichtigen Versionen der SaaS- und OnPremise-Lösung jeweils kostenlose Varianten an, welche in dieser Arbeit zur Evaluierung herangezogen wurden. Splunk selber bietet keine JavaScript-Pakete an, um Daten von einer Webanwendung an Splunk zu senden. Hingegen wird aber eine REST-Schnittstelle angeboten, um eigene Datensätze in Splunk einzufügen.

Über diese einer-Schnittstelle wurden aus der Webanwendung einerseits Loginformationen gesendet, aber auch Fehlerdaten, wie man sie z. B. bei Sentry finden konnte. Die Daten ließen sich effizient in Splunk manuell durchsuchen, filtern, und visualisieren. Dadurch dass Splunk als dieselbe Datensinke für Fehler- und Loginformationen agierte, erlaubte die Kombination aus diesen beiden Datenkategorien. Speziell heißt dies, dass man bei Fehlerfällen direkt zu den Loginformationen zur selben Sitzung wechseln konnte.

Jedoch ist das Error-Monitoring nicht vergleichbar gut wie das von Sentry, da die Funktionalität des Issue-Managements nicht vorhanden ist. Weiterhin wurde evaluiert, ob auch Tracingdaten in Splunk aufgenommen werden können, aber hierbei besitzt Splunk keine gängigen Visualisierungen, wie ein Trace-Gantt-Diagramme.

#### 3.3.8 Honeycomb

Honeycomb [Hou20] ist eine SaaS-Lösung der Hound Technology Inc. und verspricht die Speicherung vieler (Tracing-)Daten und basierend darauf effiziente Abfragen zu ermöglichen. Es ist hauptsächlich als Tracingdienst anzusehen, womit jedoch auch Aspekte des APM, RUM und Error Monitoring mit abgebildet werden können.

Honeycomb wurde mit der kostenlosen Version evaluiert. Honeycomb bietet sog. „Beelines“ an, welche Werkzeuge zur automatischen Datenerfassung sind. Diese Beelines sind für Node.js, Go, Python, Java, Ruby und Rails verfügbar. Da keine Beelines für JavaScript in Browsern verfügbar sind, wurden die Daten mit OTEL-Komponenten erhoben und ein eigener Exporter für Honeycomb entwickelt.

Honeycomb konzentriert sich auf die Datenkategorien Tracing und Metriken, ähnlich wie der Standard OpenTelemetry. Dabei bietet Honeycomb in der Oberfläche Möglichkeiten selber Graphen auf Basis der Daten zu erstellen. Somit ist dies keine ausgefertigte Lösung, wie es teils bei New Relic und Dynatrace zu finden ist und kann somit auf die Anforderungen des Projektes besser angepasst werden. Jedoch ist dieser Ansatz mit einem höheren Aufwand und einer notwendigen Expertise verbunden.

#### 3.3.9 Jaeger

Jaeger wurde 2017 als ein OpenSource-Projekt der CNCF gestartet [Jae20]. Es ist ein System für verteiltes Tracing und bietet Funktionalitäten zur Datensammlung, -verarbeitung, -speicherung bis hin zur Visualisierung. Jaeger unterstützt und implementiert den Standard OpenTracing, unterstützt aber auch Datenformate anderer Hersteller (wie z. B. Zipkin [Zip21]). Eine Unterstützung des OpenTelemetry Standards ist derzeit im Gange. Weiterhin kann Jaeger dazu benutzt werden, Metriken nach Prometheus [Pro20] zu exportieren, einem weiteren CNCF Projekt zur Speicherung und Visualisierung von Daten.

Jaeger spezialisiert sich auf Tracing und bietet hierfür eine skalierbare Infrastruktur zur Speicherung und Analyse der Daten. Die Traces werden als angereicherte Trace-Gantt-Diagramme dargestellt, wie in Abbildung 3.9 zu sehen ist. Hierbei sind sowohl hierarchische

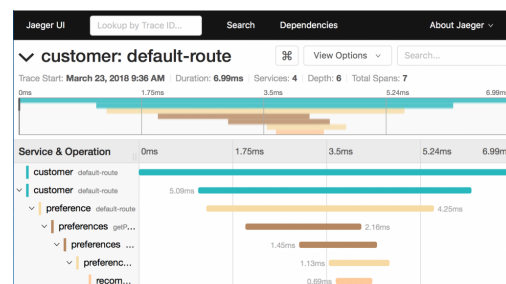


Abb. 3.9: Trace-Detailansicht. Quelle: Don Schenck [Sch20]

als auch zeitliche Beziehungen visualisiert. Wie bei OpenTracing und OpenTelemetry besteht ein Trace aus mehreren Spans, welche meist eine Methode umschließen. Zu den einzelnen Spans lassen sich weitere Informationen anzeigen, wenn vorhanden, wie bspw. Logmeldungen oder Kontextinformationen.

Anhand der Traces generiert Jaeger zudem automatisch eine Architektur, indem die Beziehungen zwischen Diensten zu sehen ist. In Abbildung 3.10 kann so eine Darstellung betrachtet werden.

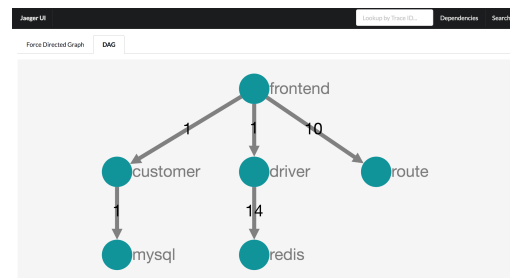


Abb. 3.10: Dienst-Abhängigkeits-Graph. Quelle: Yuri Shkuro [Shk20]

#### 3.3.10 TraVista

Passend zu Jaeger konnte in der Literatur ein Werkzeug identifiziert werden, welches auf bestehende Visualisierungen mittels Trace-Gantt-Diagrammen aufsetzt und diese erweitert. Anand *et al.* [ASD<sup>+</sup>20] argumentieren in ihrem Bericht, dass Visualisierungen rund ums Tracing zu strikt die unterschiedlichen Daten einer Anwendung trennen, statt diese zu kombinieren und strukturiert zu veranschaulichen. Mit TraVista haben die Autoren Visualisierungen erstellt, die genau diese Verknüpfung der unterschiedlichen Datentypen versuchen.

In der Oberfläche von TraVista werden Metriken, Events sowie Tracedaten simultan dargestellt, um dem Nutzer ein komplementäres Bild des eigentlichen Traceverlaufs zu präsentieren. Diese Visualisierung kann in Abbildung 3.12 betrachtet werden, und ein Zoom ist bei Abbildung 3.11 zu finden. Das Trace-Gantt-Diagramm wurde u. A. bei ② um einige Metriken erweitert, wobei der aktuelle Trace hervorgehoben im Vergleich zu anderen Traces dargestellt wird. Weiterhin lassen sich im Zoom bei ⑦ aufgetretene Events als Balken betrachten, wobei selten aufgetretene Events schwarz hervorgehoben werden.

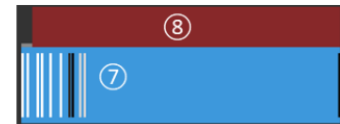


Abb. 3.11: Zoom von Abbildung 3.12, Abbildung aus [ASD<sup>+</sup>20]

Leider ist TraVista keine ausgereifte Software und wird seit einigen Monaten nicht mehr weiterentwickelt<sup>4</sup>. Aufgrund dessen wird es in dieser Arbeit nicht verwendet, dennoch zeigt sich durch den Bericht, dass bestehende und etablierte Produkte wie Jaeger und Zipkin ein Verbesserungspotenzial aufweisen.

<sup>4</sup>TraVista auf GitHub: <https://github.com/vaastav/TraViz>

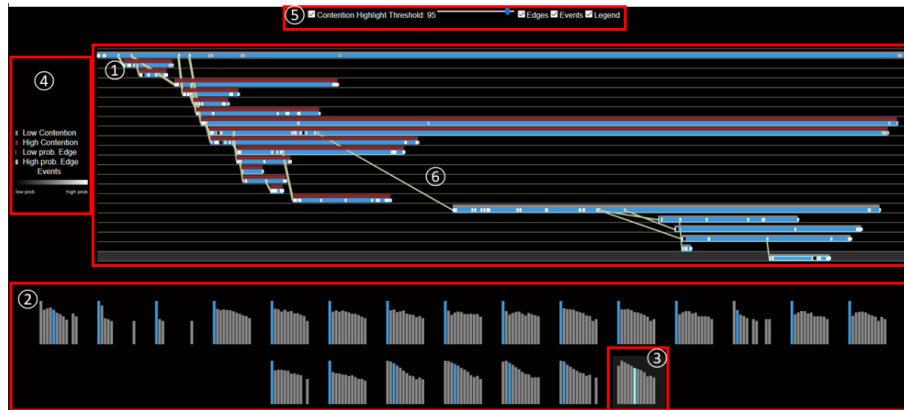


Abb. 3.12: TraVistas Gantt-Diagramm, Abbildung aus [ASD<sup>+</sup>20]

#### 3.3.11 FAME

2018 arbeiteten Oriol *et al.* [OS<sup>+</sup>18] mit der Firma SEnerCon GmbH zusammen, um ein Framework zu erstellen, welches Daten aus Nutzerfeedback und Monitoring kombiniert. SEnerCon wurde hinzugezogen, um das Framework zu evaluieren, die Entwicklung des Frameworks fand durch die Forscher selbst statt. Das Framework wurde FAME<sup>5</sup> genannt und steht für „Feedback Acquisition and Monitoring Enabler“.

In dem Bericht wurde erforscht, ob die kombinierten Daten aus Nutzerfeedback und Monitoring dabei helfen können, um neue Anforderungen zu ergründen. Auf Basis dieser Forschungsfrage wurde das Framework erstellt und es konnte am Ende ein Mehrwert für die SEnerCon GmbH identifiziert werden.

#### 3.3.12 The Kaiju Project

Scrocca *et al.* [STM<sup>+</sup>20] identifizierten 2020 eine Lücke im Gebiet der Observability, nämlich fehlt ein Werkzeug, welches die unterschiedlichen Datentypen Logs, Traces und Metriken kombiniert sammeln und aufbereiten kann und dies in quasi-Echtzeit. Aus diesem Mangel ergab sich die Forschungsfrage, ob eine Kombination dieser Datenkategorien einen Mehrwert für die Nachvollziehbarkeit bedeuten kann.

Um die Forschungsfrage zu beantworten wurde ein Prototyp konzipiert, welcher genau diese Datenkategorien aggregiert und diese zu High-Level-Events zusammenfasst. Bei der Evaluierung mit der IT Dienstleistungsfirma SighUp konnte festgestellt werden, dass die

<sup>5</sup>FAME auf GitHub: [https://github.com/supersede-project/monitor\\_feedback](https://github.com/supersede-project/monitor_feedback)

resultierenden High-Level-Events hilfreiche Informationen zur Problemfindung beinhalten.

Das Projekt Kaiju hat bisher nicht die Phase des Prototypen verlassen und wird seit einigen Monaten auch nicht weiterentwickelt<sup>6</sup>. Jedoch ist die Erkenntnis, dass die Kombination von Daten unterschiedlicher Quellen und Kategorien einen Mehrwert bietet, eine wegbereitende für diese Arbeit und die angestrebte Lösung. Somit wird in der hier angestrebten Lösung auch versucht unterschiedliche Kategorien miteinander zu verknüpfen, um Synergieeffekte für die Betreiber und Entwickler zu erzeugen. Folgend wird mit der Konzeption und Erstellung des Proof-of-Conceptes fortgesetzt.

---

<sup>6</sup>Kaiju auf GitHub: <https://github.com/marioscrock/Kaiju>



## 4 Erstellung Proof-of-Concept

### 4.1 Anforderungen

Das zu erstellende Proof-of-Concept soll einige Rahmenbedingungen erfüllen. In diesem Abschnitt werden diese Bedingungen näher beschrieben.

#### 4.1.1 Definitionen

Um die Anforderungen systematisch einzuordnen, werden sie auf Basis von zwei Modellen kategorisiert, welche folgend vorgestellt werden.

Beim ersten Modell handelt es sich um das Kano-Modell [Kan68] der Kundenzufriedenheit, welches in Tabelle 4.1 erläutert wird.

Kürzel	Titel	Beschreibung
<b>B</b>	Basismerkmal	Merkmale, die als selbstverständlich angesehen werden. Eine Erfüllung erhöht kaum die Zufriedenheit, jedoch eine Nichterfüllung führt zu starker Unzufriedenheit
<b>L</b>	Leistungsmerkmal	Merkmale, die der Kunde erwartet und bei nicht Vorhandensein in Unzufriedenheit äußert. Ein Vorhandensein erzeugt Zufriedenheit, beim Übertreffen umso mehr.
<b>S</b>	Begeisterungsmerkmal	Merkmale, die eine Herabsetzung von der Konkurrenz ermöglichen und die den Nutzenfaktor steigern. Sind sie vorhanden, steigern sie die Zufriedenheit merklich.
<b>U</b>	Unerhebliches Merkmal	Für den Kunden belanglos, ob vorhanden oder nicht.
<b>R</b>	Rückweisungsmerkmal	Diese Merkmale führen bei Vorhandensein zu Unzufriedenheit, sind jedoch beim Fehlen unerheblich.

Tab. 4.1: Merkmale nach dem Kano-Modell der Kundenzufriedenheit

Neben der Unterscheidung nach dem Kano-Modell werden die Anforderungen in funktionale und nicht-funktionale Anforderungen [Bra16] aufgeteilt (vgl. Tabelle 4.2).

Kürzel	Titel	Beschreibung
f	funktional	Beschreiben Anforderungen, welche ein Produkt ausmachen und von anderen differenzieren („Was soll das Produkt können?“). Sie sind sehr spezifisch für das jeweilige Produkt. Ein Beispiel: Das Frontend fragt Daten für X vom Partnersystem 1 über eine SOAP-API ab, etc.
nf	nicht-funktional	Beschreiben Leistungs- und Qualitätsanforderungen und Randbedingungen („Wie soll das Produkt sich verhalten?“). Sie sind meist unspezifisch und in gleicher Form auch in unterschiedlichsten Produkten vorzufinden. Beispiele sind: Benutzbarkeit, Verfügbarkeit, Antwortzeit, etc. Zur Überprüfung sind oftmals messbare, vergleichbare und reproduzierbare Definitionen notwendig.

Tab. 4.2: Kategorien der Anforderungen

#### 4.1.2 Anforderungsanalyse

Die Anforderungen, welche von der zu erstellende Lösung gefordert werden, ergaben sich durch den Einfluss verschiedener Quellen. Die primäre Quelle an Anforderungen stellen die Stakeholder dieser Arbeit, Christian Wansart und Stephan Müller, dar. Als Stakeholder betreuen sie die Arbeit und haben ein eigenes Interesse, dass aus der Arbeit ein erfolgreiches und übertragbares Ergebnis resultiert.

Neben den Stakeholdern ergeben sich auch Anforderungen direkt aus der Forschungsfrage selbst und den Bestrebungen des Autors. Die Quellen werden in den Anforderungen mit einem Kürzel angegeben, wie z. B. A für Autor, zu sehen in Tabelle 4.3.

Eine dritte Quelle von Anforderungen ergibt sich aus der Problemstellung des Kunden der Open Knowledge, welche in der Motivation angesprochen wurde. Die beiden Stakeholder brachten neben ihren eigenen Bestrebungen auch die Rahmenbedingungen und Wünsche des Kunden mit ein. Aus dieser Kommunikation ergaben sich somit weitere Anforderungen, welche einen realitätsnahen Charakter haben.

Anforderungen können auch eine Kombination von mehreren Quellen besitzen, wenn die Anforderung aus einer gemeinsamen Bestrebung oder Diskussion entstand.

Kürzel	Titel	Beschreibung
A	Autor	Hiermit ist der Autor dieser Arbeit gemeint.
S	Stakeholder	Die beiden Stakeholder Christian Wansart und Stephan Müller
K	Kunde	Ein Kunde der Open Knowledge, ein Direktversicherer.

Tab. 4.3: Quellen der Anforderungen

### 4.1.3 Anforderungsliste

Um die Anforderungen strukturiert zu erfassen, werden sie ähnlich einer Karteikarte, wie in Tabelle 4.4 zu sehen, dargestellt. Hierbei erhält jede Anforderung eine Kategorisierung nach dem Kano-Modell, ob sie funktional oder nicht-funktional ist und aus welcher Anforderungsquelle sie entstammt. Jede Anforderung erhält zudem eine eindeutige Id, die nachfolgend in der Arbeit zur Referenzierung dient.

Id	Name	Kano-Modell	Funktionsart	Quelle
1234	Dummy	S	nf	S
Hier wird die Anforderung beschrieben.				

Tab. 4.4: Beispiel einer Anforderung

#### 4.1.3.1 Grundanforderungen

Id	Name	Kano-Modell	Funktionsart	Quelle
1010	Konzept	B	f	A
Es wird ein System konzipiert, welches darauf abzielt die Nachvollziehbarkeit einer JavaScript-basierten Webanwendung zu verbessern. Speziell sollen Benutzerinteraktionen und Anwendungsverhalten nachvollziehbarer gemacht werden.				

Id	Name	Kano-Modell	Funktionsart	Quelle
1020	Demoanwendung	B	f	A
Eine Demoanwendung ist zu Erstellen und soll dazu dienen, das Konzept darauf anwenden zu können. Diese Demoanwendung soll Fehlerverhalten beinhalten, die dann mithilfe der Lösung besser nachvollziehbar zu gestalten sind.				

Id	Name	Kano-Modell	Funktionsart	Quelle
1030	Proof-of-Concept	B	f	A
Auf Basis des Konzeptes, ist die Demo-Anwendung zu erweitern.				

Id	Name	Kano-Modell	Funktionsart	Quelle
1031	Bewertung Proof-of-Concept	B	f	A
Nach Abschluss der Implementierung des Proof-of-Concepts soll dieser veranschaulicht und bewertet werden. Grundlage hierfür sind diese Anforderungen sowie die, zu identifizierende, Fähigkeit die Fehlerszenarien der Demoanwendung nachvollziehbar zu gestalten.				

#### 4.1.3.2 Funktionsumfang

Id	Name	Kano-Modell	Funktionsart	Quelle
2010	Schnittstellen-Logging	B	f	S
Das Aufrufen von Schnittstellen ist mittels einer Logmeldung zu notieren. Hierbei sind relevante Informationen wie Aufrufparameter ebenfalls zu notieren.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2011	Use-Case-Logging	B	f	S
Tritt ein Use-Case auf, soll dieser im Log notiert werden. Beispielsweise soll notiert werden, wenn ein Nutzer ein Formular absendet.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2020	Error-Monitoring	B	f	S
Tritt ein Fehler auf, der nicht gefangen wurde, so ist dieser automatisch zu erfasst und um weitere Attribute zu ergänzen. Sonstige Fehler können auch erfasst werden, aber hierbei ist keine automatische Erfassung gefordert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2030	Tracing	B	f	S
Es werden Tracingdaten ähnlich wie bei OpenTracing und OpenTelemetry erfasst. Optimalerweise werden die Tracingdaten mit OpenTelemetry-konformen Komponenten erfasst.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2040	Metriken	B	f	S
Es werden Metrikdaten ähnlich wie bei OpenTelemetry erfasst. Optimalerweise werden die Tracingdaten mit OpenTelemetry-konformen Komponenten erfasst.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2050	Session-Replay	B	f	S
Es sollen Session-Replay-Daten erhoben werden, anhand dessen die Benutzerinteraktionen und das Anwendungsverhalten nachgestellt werden kann. Diese Funktionalität darf jedoch standardmäßig deaktiviert sein.				

#### 4 Erstellung Proof-of-Concept

Id	Name	Kano-Modell	Funktionsart	Quelle
2110	Übertragung von Logs	B	f	S
Ausgewählte Logmeldungen sind an ein Partnersystem weiterzuleiten. Die Auswahl könnte über die Kritikalität, also dem Log-Level, der Logmeldung erfolgen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2120	Übertragung von Fehlern	B	f	S
Sämtlich erfasste Fehler sind an ein Partnersystem weiterzuleiten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2130	Übertragung von Tracingdaten	B	f	S
Sämtlich erfasste Tracingdaten sind an ein Partnersystem weiterzuleiten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2140	Übertragung von Metrikdaten	B	f	S
Sämtlich erfasste Metrikdaten sind an ein Partnersystem weiterzuleiten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2150	Übertragung von Session-Replay-Daten	B	f	S
Sämtlich erfasste Session-Replay-Daten sind an ein Partnersystem weiterzuleiten.				

##### 4.1.3.3 Eigenschaften

Id	Name	Kano-Modell	Funktionsart	Quelle
3010	Resilienz der Übertragung	S	f	S
Daten, die der Nachvollziehbarkeit dienen, sollen, wenn möglich, bei einer fehlgeschlagenen Verbindung nicht verworfen werden. Sie sind mindestens 120s vorzuhalten und in dieser Zeit sind wiederholt Verbindungsversuche zu unternehmen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
3020	Batchverarbeitung	S	f	S
Daten, die der Nachvollziehbarkeit dienen, sind, wenn möglich, gruppiert an externe Systeme zu senden. Hierbei ist eine kurze Aggregationszeit von bis zu 10s akzeptabel.				

Id	Name	Kano-Modell	Funktionsart	Quelle
3100	Anzahl Partnersysteme	B	nf	K
Die Anzahl an zusätzlichen Partnersystemen, die für die Lösung benötigt werden, ist so gering zu halten wie möglich.				

Id	Name	Kano-Modell	Funktionsart	Quelle
3200	Structured Logging	L	f	A+S
Das Logging soll mit einem vordefinierten Format durchgeführt werden. Für ähnliche Funktionsgruppen (wie ein Schnittstellenaufruf) soll das gleiche Format verwendet werden. Ein anwendungsübergreifendes Format ist nicht gefordert.				

#### 4.1.3.4 Partnersysteme

Id	Name	Kano-Modell	Funktionsart	Quelle
5010	Partnersystem <i>Log-Management</i>	B	f	A+S
Es existiert ein Partnersystem, zu dem Logmeldungen weitergeleitet werden. Dieses System soll die Logmeldungen speichern und den Entwicklern und Betreibern eine Einsicht in die erfassten Logmeldungen bieten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5020	Partnersystem <i>Error-Monitoring</i>	B	f	A+S
Es existiert ein Partnersystem, zu dem Fehler weitergeleitet werden. Dieses System soll die Fehler speichern und den Entwicklern und Betreibern eine Einsicht in die erfassten Fehler bieten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5021	Visualisierung <i>Error-Monitoring</i>	L	f	A+S
Das Partnersystem, zu dem die Fehler weiterzuleiten sind, soll diese grafisch darstellen können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5022	Alerting <i>Error-Monitoring</i>	S	f	A+S
Das Partnersystem, zu dem die Fehler weiterzuleiten sind, soll bei Auftreten von bestimmten Fehlern oder Fehleranzahlen eine Meldung erzeugen (per E-Mail, Slack, o. Ä.).				

Id	Name	Kano-Modell	Funktionsart	Quelle
5030	Partnersystem <i>Tracing</i>	B	f	A+S
Es existiert ein Partnersystem, zu dem Tracingdaten weitergeleitet werden. Dieses System soll die Fehler speichern und den Entwicklern und Betreibern eine Einsicht in die erfassten Tracingdaten bieten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5031	Visualisierung <i>Tracing</i>	B	f	A+S
Das Partnersystem, zu dem die Tracingdaten weitergeleitet werden, soll diese grafisch als Tracing-Wasserfallgraph darstellen können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5040	Partnersystem <i>Metriken</i>	L	f	A+S
Es existiert ein Partnersystem, zu dem Metriken weitergeleitet werden. Dieses System soll die Fehler speichern und den Entwicklern und Betreibern eine Einsicht in die erfassten Metriken bieten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5041	Visualisierung <i>Metriken</i>	L	f	A+S
Das Partnersystem, zu dem die Metriken weiterzuleiten sind, soll diese grafisch darstellen können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5042	Alerting <i>Metriken</i>	S	f	A+S
Das Partnersystem, zu dem die Metriken weiterzuleiten sind, soll bei Auftreten von bestimmten Metrikwerten oder Überschreitungen von Schwellen eine Meldung erzeugen (per E-Mail, Slack, o. Ä.).				

Id	Name	Kano-Modell	Funktionsart	Quelle
5050	Partnersystem <i>Session-Replay</i>	B	f	A+S
Es existiert ein Partnersystem, zu dem die Session-Replay-Daten weitergeleitet werden. Dieses System soll anhand dieser Daten eine Benutzersitzung rekreieren können.				

## 4.2 Vorstellung der Demoanwendung

Es wird eine Demoanwendung erstellt, welche die Konzepte anhand der Anforderung 1020 umsetzt. Dieser Abschnitt beschäftigt sich mit der Vorstellung der Demoanwendung und

der repräsentativen Aufgabe, die diese übernimmt.

In der Motivation wurde ein konkretes Problem eines Kunden der Open Knowledge genannt. Damit die Demoanwendung realistisch eine moderne Webanwendung darstellt, wird sie in Grundzügen den Aufbau der Webanwendung des Direktversicherers nachahmen. Bei der Webanwendung handelt es sich um einen Wizard, also einer Sequenz von aufeinanderfolgenden Dialogseiten bei dem der Nutzer Daten eingeben soll. Bei der Webanwendung handelt es sich um eine clientbasierte Angular-SPA. Die Webanwendung validiert einzelne Felder gegen Partnersysteme (bspw. beim Adressfeld). Am Ende des Wizards werden die gesamten Daten an ein weiteres Partnersystem übermittelt, welches darauf basierend eine Berechnung durchführt und das Ergebnis dann an die Webanwendung sendet.

Es wurde sich dafür entschieden, dass die Webanwendung eine Bestellfunktionalität eines Obst-Webshops darstellen soll. Der Warenkorb hierfür wird anfangs dynamisch generiert und dies soll so simulieren, dass eine andere Komponente diesen erstellt hat. Der Nutzer soll seine Rechnungs- und Lieferdaten eingeben und am Ende die Bestellung ausführen können. Um das gewünschte Verhalten der Demoanwendung zu definieren, wird es im folgenden Abschnitt festgelegt.

### 4.2.1 Verhaltensdefinition

Mit den beiden Stakeholdern, also Christian Wansart und Stephan Müller, die beide am Projekt für den Kunden involviert sind, wurde diese Verhaltensdefinition erstellt. Diesen Ansatz der Definition der Software anhand des Verhaltens nennt man Behavior-Driven Development (BDD). BDD wurde 2006 erstmals von Dan North benannt und definiert [Nor06]. Bei BDD werden User-Stories aus der Sicht eines äußerlichen Betrachters entworfen und geschrieben. Dabei umfassen die User-Stories Beispiele, wie sich die Anwendung in diesen Szenarien verhalten soll.

Um die BDD-Definition festzuhalten wurde sie in der gängigen Gherkin-Syntax [Sma19] geschrieben. Die Syntax ist natürlich zu lesen, folgend werden alle gewünschten Features der Demoanwendung in der Gherkin-Syntax aufgelistet.

```
1 Feature: Warenkorb
2
3     Der Warenkorb ist eine Übersicht über die gewählten Artikel. Hier
        sollen die Artikel samt Name, Anzahl sowie Preis angezeigt werden.
        Der Warenkorb stellt den Einstieg der Software dar.
4
5 Scenario: Kundin öffnet den Warenkorb
6     When die Kundin den Warenkorb öffnet
7     Then soll sie die ausgewählten Artikel mit Bild, Artikelnamen,
        Anzahl und dem Gesamtpreis des Artikels sehen
8     And sie soll den Gesamtpreis für alle Artikel sehen
```



```
9
10 Scenario: Kundin soll zur nächsten Seite wechseln können
11 Given die Kundin hat die gewählten Produkte geprüft
12 When sie auf den "Bestellvorgang starten"-Button klickt
13 Then soll sie auf die Seite "Rechnungsadresse" gelangen
```

Quellcode 4.1: Demoanwendung: Gherkin Definition zum Feature „Warenkorb“

```
1 Feature: Rechnungsadresse
2
3 Die zweite Seite ist die Rechnungsadresse. Hier sollen die Nutzer ihre
  Rechnungsadresse eingeben können, welche die Pflichtfelder Anrede
  , Vornamen, Nachnamen, Straße, Hausnummer, Postleitzahl sowie die
  E-Mail-Adresse umfassen.
4
5 Scenario: Kundin kommt auf die Rechnungsadresse-Seite vom Warenkorb
  aus
6 When die Rechnungsadresse-Seite zum ersten Mal aufgerufen wird
7 Then sollen die Eingabefelder leer sein
8
9 Scenario: Kundin kommt auf die Rechnungsadresse-Seite von der
  Lieferadresse-Seite aus
10 Given die Kundin hatte bereits zuvor die Rechnungsadresse ausgefü
    llt
11 Then sollen die zuvor eingegebenen Adressdaten weiterhin vorhanden
    sein
12
13 Scenario: Kundin kann ihre Rechnungsadresse eingeben
14 When die Kundin die Rechnungsadresse-Seite betritt
15 Then soll sie die Möglichkeit haben
16 * eine Anrede anzugeben
17 * den Vornamen eingeben zu können
18 * den Nachnamen eingeben zu können
19 * die Straße eingeben zu können
20 * die Hausnummer eingeben zu können
21 * die Postleitzahl (PLZ) eingeben zu können
22 * die Stadt eingeben zu können
23 * die E-Mail-Adresse eingeben zu können
24
25 Scenario: Kundin soll zur nächsten Seite wechseln können
26 Given die Kundin hat alle Felder ausgefüllt
27 When sie auf den "weiter"-Button klickt
28 Then soll sie auf die Seite "Lieferdaten" gelangen
29
30 Scenario: Kundin füllt nicht alle benötigten Felder aus und klickt auf
    "weiter"
31 Given die Kundin hat alle Felder außer bspw. der Hausnummer
    eingegeben
32 When sie auf "weiter" klickt
33 Then soll sie informiert werden, dass sie alle Felder ausfüllen
    muss
```

```
34
35 Scenario: Kundin gibt invalide Daten ein
36 When die Kundin eine andere Rechnungsadresse eingibt
37 * Vorname und Nachname Sonderzeichen enthalten außer Bindestriche
   enthält
38 * Straße Sonderzeichen außer Bindestriche und Punkte enthält
39 * Hausnummer Sonderzeichen enthält
40 * PLZ alles andere außer Zahlen enthält
41 * Stadt keine deutsche Stadt ist
42 * das @ bei der E-Mail-Adresse fehlt
43 Then soll eine Warnung angezeigt werden und der "weiter"-Button
   blockiert werden
```

Quellcode 4.2: Demoanwendung: Gherkin Definition zum Feature „Rechnungsadresse“

```
1 Feature: Lieferdaten
2
3   Auf der dritten Seite sollen die Kunden die Lieferdaten eintragen kö
   nnen. Hier soll es die Möglichkeit geben, die Rechnungsadresse als
   Lieferadresse übernehmen zu können. Alternativ sollen die Nutzer
   die Pflichtfelder Anrede, Vornamen, Nachnamen, Straße, Hausnummer,
   Postleitzahl, Stadt eingeben können.
4
5 Scenario: Kundin kommt auf die Lieferdaten-Seite von der
   Rechnungsadresse-Seite aus
6 When die Kundin zum ersten Mal auf die Lieferdaten-Seite kommt
7 Then soll das Häkchen bei "Gleiche Lieferdaten wie
   Rechnungsadresse" gesetzt sein
8 And das gleiche Formular wie von der Rechnungsadresse-Seite
   erscheinen, mit den zuvor eingegebenen Daten
9 And das Formular soll deaktiviert sein, solange das Häkchen
   gesetzt ist
10
11 Scenario: Kundin kommt auf die Lieferdaten-Seite von der Zahlungsdaten
   -Seite aus
12 Given die Kundin hatte bereits zuvor die Lieferdaten ausgefüllt
13 Then sollen die zuvor eingegebenen Adressdaten weiterhin vorhanden
   sein
14
15 Scenario: Kundin möchte die Rechnungsadresse übernehmen
16 Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"
   ist gesetzt
17 When sie auf den "weiter"-Button klickt
18 Then soll sie auf die Seite "Zahlungsdaten" gelangen
19
20 Scenario: Kundin möchte andere Lieferdaten nutzen
21 Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"
   wurde entfernt
22 When die Kundin hat alle Felder ausgefüllt
23 And sie auf den "weiter"-Button klickt
24 Then soll sie auf die Seite "Zahlungsdaten" gelangen
```

```
25
26 Scenario: Kundin möchte andere Lieferdaten nutzen, ohne alle Felder
    ausgefüllt zu haben
27 Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"
    wurde entfernt
28 When die Kundin eine andere Lieferdaten eingibt
29 * Vorname und Nachname Sonderzeichen enthalten außer Bindestriche
    enthält
30 * Straße Sonderzeichen außer Bindestriche und Punkte enthält
31 * Hausnummer. Sonderzeichen enthält
32 * PLZ alles andere außer Zahlen enthält
33 * Stadt keine deutsche Stadt ist
34 * das @ bei der E-Mail-Adresse fehlt
35 And sie auf den "weiter"-Button klickt
36 Then soll eine Warnung angezeigt und der "weiter"-Button blockiert
    werden
```

Quellcode 4.3: Demoanwendung: Gherkin Definition zum Feature „Lieferadresse“

```
1 Feature: Zahlungsart
2
3 Die vierte Seite enthält die Auswahl der Zahlungsart. Hier sollen den
    Kunden die Zahlungsarten Rechnung, Lastschrift, PayPal und
    Kreditkarte zur Auswahl gestellt werden.
4
5 Scenario: Kundin kommt zum ersten Mal auf die Zahlungsdaten-Seite von
    der Lieferdaten-Seite
6 When die Kundin die Seite zum ersten Mal betritt
7 Then soll "Rechnung" vorausgewählt sein
8
9 Scenario: Kundin kommt auf die Zahlungsdaten-Seite von der "Bestellung
    abschließen"-Seite aus
10 Given die Kundin hatte bereits zuvor die Zahlungsart ausgefüllt
11 Then sollen die zuvor eingegebenen Zahlungsdaten weiterhin
    vorhanden sein
```

Quellcode 4.4: Demoanwendung: Gherkin Definition zum Feature „Zahlungsdaten“

```
1 Feature: Bestellung abschließen
2
3 Die letzte Seite soll eine Übersicht über die zuvor eingegebenen Daten
    geben, bevor die Kundin die Bestellung abschließt.
4
5 Scenario: Kundin betritt die Seite
6 When die Kundin die Seite betritt soll eine Bestellübersicht über
    die Artikel von Seite 1 angezeigt werden
7 * die Artikel angezeigt werden
8 * die Rechnungsadresse angezeigt werden
9 * die Lieferadresse angezeigt werden
10 * die Rechnungsart angezeigt werden
11 * ein "kostenpflichtig bestellen"-Button angezeigt werden
```

```
12
13   Scenario: Kundin schließt die Bestellung ab
14   When die Kundin auf den "kostenpflichtig bestellen"-Button klickt
15   Then soll eine Serverinteraktion ausgelöst werden, die die
      Bestellung speichert
16   And die Bestellbestätigung soll dargestellt werden
```

Quellcode 4.5: Demoanwendung: Gherkin Definition zum Feature „Bestellung abschließen“

Neben dem eigentlichen User-Interface soll auch ein Backend Teil der Demoanwendung sein. Hierfür wurde auf Basis der Verhaltensdefinition eine Architektur entworfen, die im folgenden Abschnitt näher beschrieben wird.

### 4.2.2 Backend

Das Backend wurde als Microservice-Architektur [NMMA16] konzipiert und wurde ebenso wie die Webanwendung auch an das Projekt des Open Knowledge Kunden angelehnt. In Abbildung 4.1 lässt sich die konzipierte und umgesetzte Architektur betrachten, hierbei stellen Pods einzelne Containersysteme dar. Diese Architektur wurde mit den Stakeholdern zusammen konzipiert und ähnelt dem des Direktversicherers.

Für das Frontend ist die einzig anzusprechende Schnittstelle das „backend4frontend“, welches die Kommunikation zu Partnersystemen ermöglicht sowie die Sicherheits- und Validitätsaspekte überprüft. Die weiteren Dienste „Bestellungen“, „Übersetzungen“, „Adressvalidierung“ und „Warenkorb“ übernehmen die jeweilige Funktion, die ihr Name beschreibt. Der Dienst „Bestellungen“ ist das Partnersystem, welches beim Fertigstellen des Wizards aufgerufen wird und es führt dabei weitere Datenabfragen und Validitätsüberprüfungen mit Partnerdiensten durch.

Mit dieser recht komplexen Architektur einer Demoanwendung wurde versucht, eine möglichst realitätsnahe Repräsentation zu erstellen. Speziell wird bei einer solchen Architektur der Nutzen von Tracing deutlicher, nämlich um z. B. die Zusammenhänge zwischen den Diensten nachvollziehen zu können. Dies wird beim Einsatz und der Vorstellung der Lösung näher betrachtet.

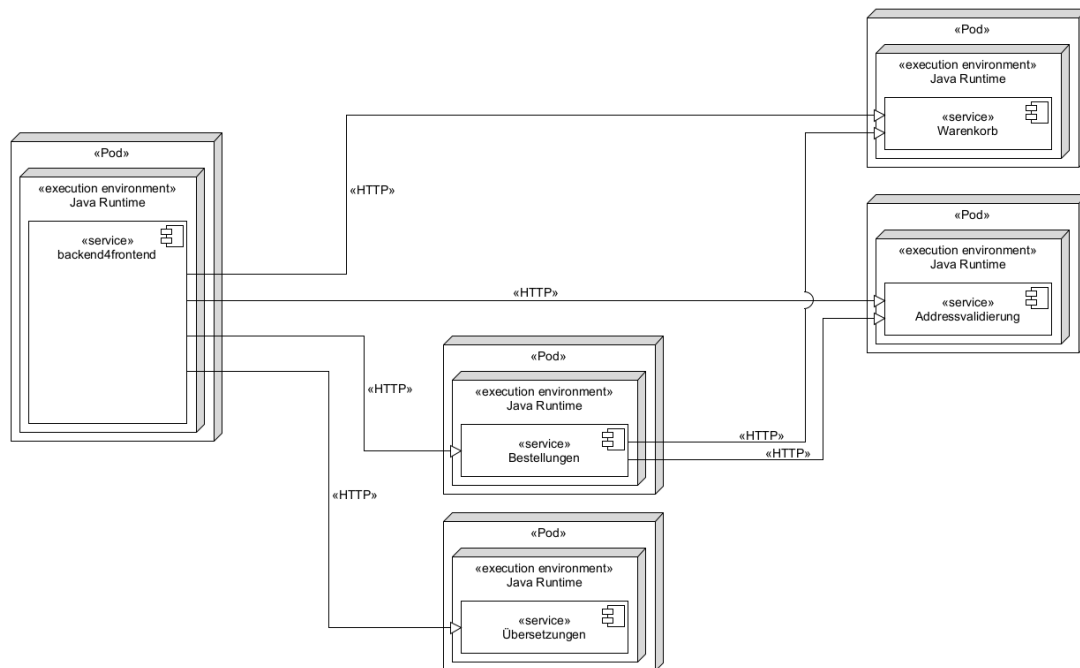


Abb. 4.1: Demoanwendung: Deployment-Diagramm, Quelle: Eigene Darstellung

Die einzelnen Dienste wurden mit Eclipse MicroProfile [Ecl21] umgesetzt. MicroProfile ist eine Kollektion von verschiedener Java EE Frameworks und Technologien zur Umsetzung von Microservices und zielt darauf ab diese weiterzuentwickeln und zu standardisieren. Speziell wurden die Dienste auf Basis von JAX-RS als REST-Services umgesetzt.

Danach wurden die einzelnen Dienste in jeweils eigene Docker Images [Doc20] verpackt, um eine einheitliche Umgebung auch auf unterschiedlichen Rechnern zu gewährleisten. Diese Images wurden dann mit Kubernetes [The20] aufgesetzt und miteinander verbunden.

### 4.2.3 Frontend

Wie beim Kunden wurde ein Wizard auf Basis von Angular erstellt, welcher mehrere aufeinander folgende Formulare in derselben SPA enthält. In den folgenden Abschnitten wird das Frontend anhand eines Beispieldurchlaufs durch die einzelnen Seiten vorgestellt.

#### 4.2.3.1 Warenkorb

Abbildung 4.2 zeigt die Startseite, die der Nutzer sieht, wenn er die Demoanwendung aufruft. Hierbei wird simuliert, dass der Nutzer zuvor in einem Online-Obsthandel einige Produkte ausgewählt hat und sich nun auf der Ansichtsseite des Warenkorbs befindet. Hier kann der Nutzer seine Auswahl prüfen und bei Zufriedenheit kann er den Bestellvorgang starten. Die hier angezeigten Daten werden vom Warenkorbdienst abgerufen, über die Angabe eines zuvor zufällig generierten Warenkorb-Identifiers. Die Warenkorbdaten werden zudem mit Übersetzungsdaten vom Übersetzungsdienst angereichert, denn in den Warenkorbdaten stehen lediglich Übersetzungsschlüssel wie `item.peach`, die dann auf den tatsächlichen Übersetzungswert abgebildet werden also `Pfirsich`. Beim Starten des Bestellvorgangs wird keine Serverabruf durchgeführt, sondern in der SPA ein Seitenwechsel vorgenommen.

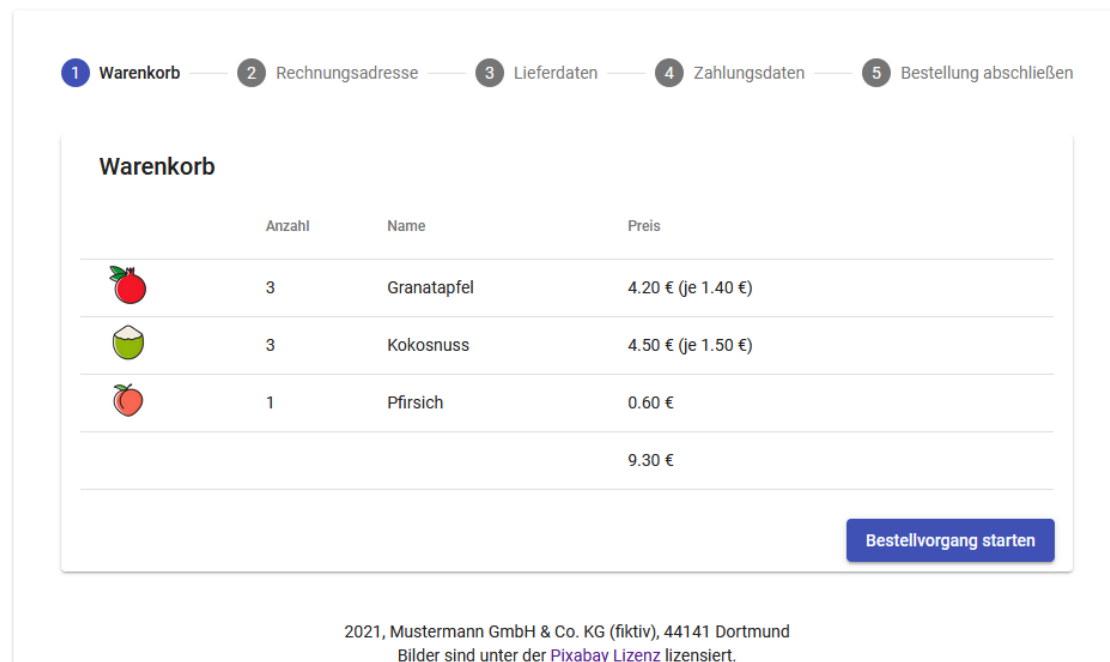


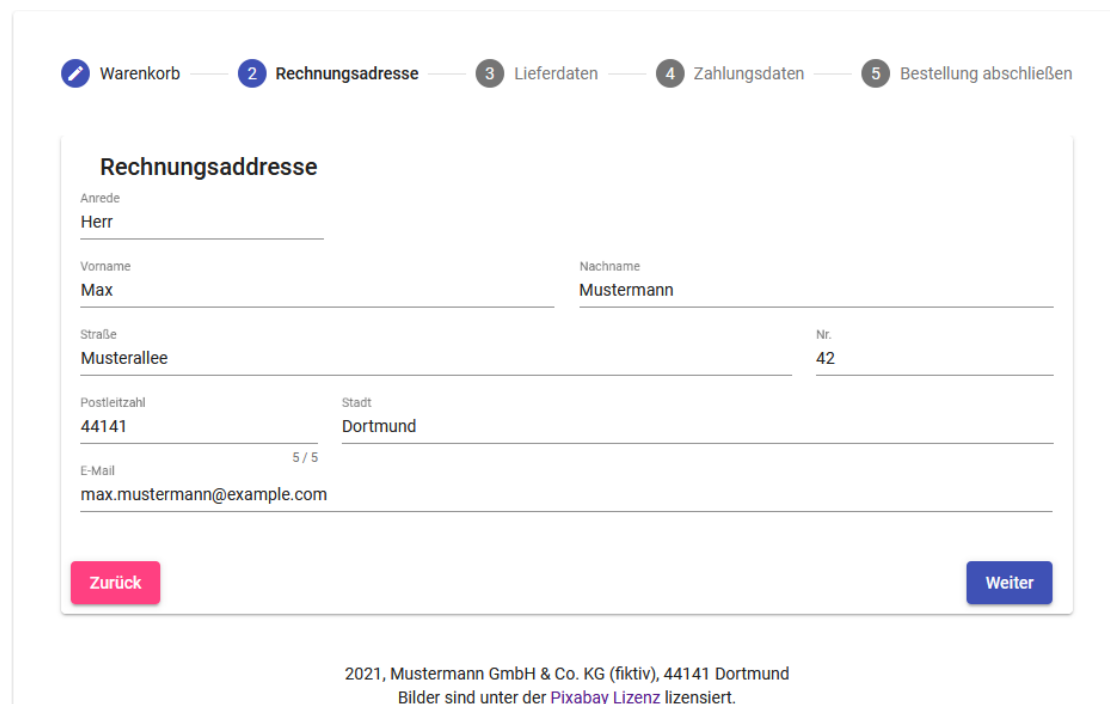
Abb. 4.2: Demoanwendung: Startseite „Warenkorb“

### 4.2.3.2 Rechnungsadresse

Startet der Nutzer den Bestellvorgang, so landet er zunächst auf der Eingabemaske zur Rechnungsadresse (vgl. Abbildung 4.3). Hier wird er gebeten rechnungsrelevante Informationen anzugeben, u. A. seine Adresse. Er kann jedoch auch auf die vorherige Seite zurückspringen.

Beim Absenden des Formulars wird zunächst die Validität der Eingabefelder überprüft, bspw. ob die PLZ aus 5 Zahlen besteht, und anschließend wird die Adresse dem Adressvalidierungsdienst zur Prüfung übergeben. Schlägt eine Validierung fehl, so wird dies entweder direkt am verursachenden Textfeld angezeigt oder in einer allgemeinen Fehlermeldung im unteren Bereich der Eingabemaske ausgegeben.

Sind beide Prüfungen jedoch erfolgreich, so wird ein Seitenwechsel in der SPA durchgeführt. Neben der Adressüberprüfung wird kein zusätzlicher Serveraufruf durchgeführt, die eingegeben Daten werden jedoch intern einer übergreifenden Komponente übergeben.



Warenkorb — 2 Rechnungsadresse — 3 Lieferdaten — 4 Zahlungsdaten — 5 Bestellung abschließen

### Rechnungsadresse

Anrede  
**Herr**

Vorname  
**Max**

Nachname  
**Mustermann**

Straße  
**Musterallee**

Nr.  
**42**

Postleitzahl  
**44141**

Stadt  
**Dortmund**

E-Mail  
**max.mustermann@example.com**

5 / 5

**Zurück** **Weiter**

2021, Mustermann GmbH & Co. KG (fiktiv), 44141 Dortmund  
Bilder sind unter der [Pixabay Lizenz](#) lizenziert.

Abb. 4.3: Demoanwendung: Seite „Rechnungsadresse“

### 4.2.3.3 Lieferdaten

Nach einer erfolgreichen Eingabe der Rechnungsadresse, wird der Nutzer nun gebeten seine Daten einzugeben, wo die Produkte hin geliefert werden sollen. Hierzu kann er entweder die relevanten Daten aus der Rechnungsübernehmen lassen (Standardfall) oder er gibt alternativ abweichende Lieferdaten an, wie in Abbildung 4.4 zu sehen ist. Wie zuvor kann der Nutzer auch auf das vorherige Formular zurückspringen.

Bei der Angabe von abweichenden Lieferdaten werden, wie beim Formular der Rechnungsadresse, zunächst die Eingabefelder überprüft und bei Fehlschlag visuell dem Nutzer darüber berichtet. Anders als bei der Rechnungsadresse wird jedoch nicht die Adressvalidierungsdienst befragt, dies wird in aufgefasset und erläutert.

Sind keine abweichenden Lieferdaten erwünscht oder die Validierung der Eingaben erfolgt, wird beim Klick auf „Weiter“ ein Seitenwechsel in der SPA durchgeführt. Auch hier erfolgt kein Serveraufruf, jedoch werden die Daten an die übergreifende Komponente innerhalb der Webanwendung weitergereicht.

Warenkorb — Rechnungsadresse — **3 Lieferdaten** — 4 Zahlungsdaten — 5 Bestellung abschließen

### Lieferdaten

☐ Gleiche Lieferadresse wie Rechnungsadresse

---

Anrede  
**Herr**

Vorname  
**Peter**

Nachname  
**Mustermann**

Straße  
**Musterring**

Nr.  
**1337**

Postleitzahl  
**44135**

Stadt  
**Dortmund**

5 / 5

**Zurück** **Weiter**

2021, Mustermann GmbH & Co. KG (fiktiv), 44141 Dortmund  
Bilder sind unter der [Pixabay Lizenz](#) lizenziert.

Abb. 4.4: Demoanwendung: Seite „Lieferdaten“



### 4.2.3.4 Zahlungsdaten

Anschließend der Eingabe der Lieferdaten wird der Nutzer nun gebeten seine Zahlungsinformationen einzugeben. Hierbei kann der Nutzer zwischen 4 Zahlungsarten auswählen: per Rechnung, Lastschrift, PayPal oder Kreditkarte. Wie bei den anderen Formularen kann der Nutzer auf das vorhergehende Formular über den Button „Zurück“ wechseln.

Bei Auswahl der Rechnungsart „Rechnung“ muss der Nutzer keine weiteren Daten eingeben. Hingegen sind bei anderen Rechnungsarten weitere Daten einzugeben, wie z. B. der in der Abbildung 4.5 zu betrachteten Rechnungsart „Lastschrift“, hier muss der Kontoinhaber und die IBAN angegeben werden. Bei PayPal muss die PayPal-E-Mail werden und bei der Auswahl der Kreditkarte müssen die Kreditkarteninformationen Karteninhaber, Kartennummer, CVC sowie das Ablaufdatum angegeben werden. Die jeweilig einzugebenden Daten werden clientseitig validiert, ähnlich wie bei den vorherigen Formularen.

Ist eine Rechnungsart ausgewählt und die einzugebenden Daten valide ausgefüllt, so führt ein Absenden des Formulars zu einem Seitenwechsel auf die Seite zum Abschließen der Bestellung. Es wird keine zusätzliche Serverinteraktion durchgeführt, die Daten werden jedoch erneut an die übergreifende Komponenten übergeben.

Abb. 4.5: Demoanwendung: Seite „Zahlungsdaten“

#### 4.2.3.5 Bestellübersicht

Da der Nutzer nun alle notwendigen Daten zur Bestellung eingegeben hat, wird auf dieser Seite ihm eine Übersicht dieser Eingaben präsentiert, wie in Abbildung 4.6 zu sehen ist. Wie bei allen Formularen gibt es auch hier die Option für den Nutzer zu einem vorherigen Formular zurückzuspringen und Anpassungen vorzunehmen.

In der Bestellübersicht werden explizit der ausgewählte Warenkorb, die eingegebenen Rechnungs- und Lieferadresse sowie die gewählte Zahlungsart dargestellt. Der Warenkorb wird hierbei analog zur Warenkorbseite vom Warenkorbdienst abgefragt und nicht clientseitig gespeichert. Die anderen Daten sind nur clientseitig gespeichert und werden über eine übergreifende Komponente bereitgestellt. Weitere Eingaben sind auf dieser Seite vom Nutzer aber nicht gefordert, sie dient hauptsächlich der visuellen Überprüfung für den Nutzer bevor er die Bestellung kostenpflichtig durchführt.

Sendet der Nutzer die Bestellung ab, werden die zuvor eingegeben Daten und der Identifier des Warenkorbs an den Bestelldienst übergeben. Dieser überprüft beide Adresseingaben gegen den Adressvalidierungsdienst, ruft den Warenkorb vom Warenkorbdienst ab und errechnet auf dieser Datenbasis den Bestellbeleg. Der Bestellbeleg wird dem Frontend in der Antwort übergeben und dies führt zu einem Seitenwechsel.

**Bestellübersicht**

**Warenkorb**

	Anzahl	Name	Preis
	3	Granatapfel	4.20 € (je 1.40 €)
	3	Kokosnuss	4.50 € (je 1.50 €)
	1	Pfirsich	0.60 €
			9.30 €

**Rechnungsadresse**  
Herr Max Mustermann  
Musterallee 42  
44141 Dortmund  
max.mustermann@example.com

**Lieferadresse**  
Herr Peter Mustermann  
Musterring 1337  
44135 Dortmund

**Zahlungsart**  
Bezahlung per Lastschrift

[Zurück](#) [Kostenpflichtig bestellen](#)

2021, Mustermann GmbH & Co. KG (fiktiv), 44141 Dortmund  
Bilder sind unter der [Pixabay Lizenz](#) lizenziert.

Abb. 4.6: Demoanwendung: Seite „Bestellübersicht“

### 4.2.3.6 Bestellbestätigung

Über die erfolgreiche Bestellung leitet die SPA automatisch auf diese Seite weiter. Hier werden die Daten der Bestellbestätigung dem Nutzer visuell präsentiert (vgl. Abbildung 4.7). Bei den angezeigten Daten handelt es sich nicht um die zuvor gespeicherten, sondern ausschließlich um die vom Bestelldienst übermittelten Daten.

Auf dieser Seite kann der Nutzer nun nur noch auf den Button „Zum Shop“ klicken und gelangt erneut zur Startseite, jedoch mit einem neuen Warenkorb.

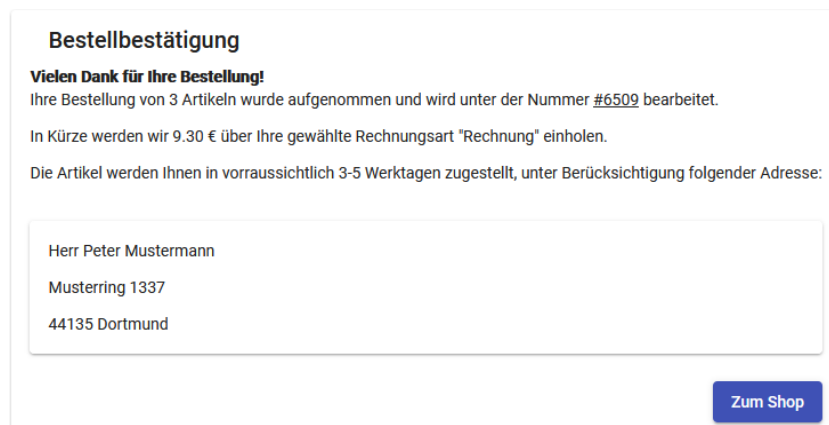


Abb. 4.7: Demoanwendung: Finale Seite „Bestellbestätigung“

Wie in Anforderung 1020 definiert wurden in das Frontend einige Fehler eingebaut, damit diese, mit der zu erstellenden Lösung, aufgedeckt werden können. Diese Fehler werden nachfolgend näher betrachtet.

### 4.2.4 Fehlerszenarien

Wie zuvor erwähnt und in Anforderung 1020 gewünscht, besitzt die Demoanwendung einige simulierte Fehler. Diese Fehler wurden in Zusammenarbeit mit den Stakeholdern konzipiert. Bei der Konzeption wurde versucht möglichst realitätsnahe oder sogar tatsächlich beim Kunden aufgetretene Probleme einzubauen.

Diese Fehler gehören unterschiedlichen Problemgruppen an, sie reichen von unerwünscht strenger Validierung, über Konfigurationsfehlern bis hin zu ineffizienter Datenverarbeitung. Sie werden folgen in Fehlerszenarien beschrieben, aus der Sicht eines Projektteams, welches diese Szenarien berichtet bekommen oder selbst notiert hat.

#### 4.2.4.1 „Keine Übersetzungen“

- Problem: Nutzer berichten, dass manchmal die Webanwendung beim Start keine Artikeltexte anzeigt (vgl. Abbildung 4.8).

- Ursache: Die Pods, die den Übersetzungsdienst enthalten, werden repliziert bereitgestellt. Einer der Pods hat eine defekte Konfiguration, weswegen er keine Übersetzungen der Artikel enthält. Wird zu diesem Pod verbunden, tritt das Fehlverhalten auf. Dies ist eine Nachstellung eines tatsächlichen Problems beim Kunden.



	Anzahl	Name
	3	***item.coconut***
	2	***item.peach***

Abb. 4.8: Fehlende Texte

#### 4.2.4.2 „Gültige Straßen sind ungültig“

- Problem: Nutzer berichten, dass Ihr Straßennamen nicht eingegeben werden kann. Beispielsweise führt die Eingabe „Ährenweg“ zu einem Fehler.

- Ursache: Der Adressvalidierungsdienst validiert Straßen mit der Regular-Expression `[a-zA-Z\,\-\ ]+`, welches keine gängigen Sonderzeichen (ä, ö, ü, ß) erlaubt.

#### 4.2.4.3 „Gültige Städte sind ungültig“

- Problem: Nutzer aus Gießen berichten, dass Sie das Formular zur Rechnungsadresse nicht ausfüllen können

- Ursache: Der Adressvalidierungsdienst meldet die Stadt „Gießen“ als ungültig, weil sie nicht in der lokalen Tabelle vorhanden ist.

#### 4.2.4.4 „Ungültige Adressen sind gültig“

- Problem: Nutzer können in den Lieferdaten ungültige Eingaben tätigen und absenden, bei der Bestellaufgabe kommt es zu einem Fehler.

- Ursache: Das Frontend überprüft lediglich die Rechnungsadresse, aber nicht die Lieferadresse

#### 4.2.4.5 „Vor- und Nachnamen werden abgeschnitten“

- Problem: Nutzer berichten, dass in der Bestellbestätigung Ihre Vor- und Nachnamen abgeschnitten dargestellt werden.
- Ursache: Der Bestelldienst begrenzt den Vor- sowie den Nachnamen auf 20 Zeichen, das Frontend begrenzt dies jedoch nicht.

#### 4.2.4.6 „Falsche Zahlungsart“

- Problem: Nutzer berichten, dass in der Bestellbestätigung die falsche Zahlungsart angezeigt wird. In der Bestellübersicht wurde jedoch die korrekte Zahlungsart angezeigt.
- Ursache: Das Frontend sendet alle Formulardaten jeder Rechnungsart an den Dienst „Bestellungen“. Dieser nimmt aber an, dass alle nicht ausgewählten Rechnungsarten statt Formulardaten nur `null` enthalten.

#### 4.2.4.7 „Lange Verarbeitung“

- Problem: Beim Absenden des Formulars auf der Seite „Warenkorb“ kommt es zu einer unerwünschten Wartezeit (von min. 6-10s).
- Ursache: Dies ist eine simulierte Wartezeit im Frontend je nach Anzahl der Positionen (2s pro Position), um eine ineffiziente Verarbeitung nachzuahmen.

Dennoch ist anzumerken, dass die Demoanwendung nur ein Modell einer tatsächlichen Webanwendung darstellt. Wie jedes Modell können nicht alle Gegebenheiten des zu modellierenden Sachverhalts nachgestellt werden. Jedoch ist durch den allgemein gehaltenen Anwendungsfall und die moderne Umsetzung eine Übertragbarkeit zu ähnlichen Projekten durchaus vorhanden.

Nun da die Demoanwendung beschrieben ist, wird das Konzept erstellt, welches letztendlich auf die Demoanwendung anzuwenden ist. Das Konzept selber ist jedoch losgelöst von der Demoanwendung zu verstehen.

## 4.3 Konzept

### 4.3.1 Datenverarbeitung

Auf Basis der zuvor vorgestellten Methoden und Praktiken wird nun eine sinnvolle Kombination für das Frontend konzeptioniert, die als Ziel hat, die Nachvollziehbarkeit nachhaltig zu erhöhen. Es werden die Grunddisziplinen Datenerhebung, -auswertung und -präsentation unterschieden und nacheinander beschrieben. Danach und darauf aufbauend wird eine grobe Architektur vorgestellt, die diese Ansätze in ein Gesamtbild bringt.

#### 4.3.1.1 Erhebung

Wie zuvor in Unterabschnitt 2.3.1 beschrieben, erhalten Betreiber und Entwickler im Normalfall nur unzureichende Information über das Anwendungsverhalten oder die getätigten Nutzerinteraktionen bei einer SPA. Aus diesem Grund sollen explizit weitere Daten erhoben werden, um die Nachvollziehbarkeit zu erhöhen.

Wie aus den Erkenntnissen von FAME (Unterabschnitt 3.3.11) und Kaiju (Unterabschnitt 3.3.12) zu deuten ist, gibt es durch die Verknüpfung von verschiedenen Datenkategorien einen Mehrwert für die Verständnis von Betreibern und Entwicklern. Deshalb sollen in der Lösung die 4 Datenkategorien „Logs“, „Metriken“, „Traces“ und „Fehler“ erhoben und an Partnersysteme weitergeleitet werden.

Neben diesen Daten sollen auch die Benutzerinteraktionen aufgezeichnet werden. Hierfür soll jedoch kein tiefer gehendes Real-User-Monitoring zur Verwendung kommen, stattdessen soll ein Session-Replay-Mechanismus eingesetzt werden. RUM wird nicht gefordert, da es für die Verständniserlangung der Benutzerinteraktionen weniger aussagekräftig ist als Session-Replay. Bei Session-Replay werden die Benutzerinteraktionen im Kontext dargestellt und nicht gesondert oder abstrahiert, was für eine Nachvollziehbarkeit hinderlich sein kann. Beim Session-Replay wird jedoch jedwede Ein- und Ausgaben der Webanwendung aufgezeichnet, damit dies nicht ein zu großes Datenvolumen erzeugt und um den Nutzer nicht konstant zu überwachen, sollte sie standardmäßig abgeschaltet sein und nur auf expliziten Nutzerwunsch aktiviert werden.

#### 4.3.1.2 Auswertung

Die genaue Auswertung ist Teil der Implementierung und diese Disziplin sieht keine direkten Vorgaben vor. Jedoch sind die gemeldeten Daten mit Kontextinformationen anzureichern, wenn möglich. Diese umfassen bspw. Zeitstempel, User-Agent, IP, Browser.

#### 4.3.1.3 Präsentation

Um ein zufriedenstellendes Ergebnis zu gewährleisten, sollte die Lösung die Daten auf folgende Weise den Betreibern und Entwicklern präsentieren:

1. Logdaten..
  - a) ..lassen sich einsehen.
  - b) ..lassen sich basierend auf ihren Eigenschaften filtern.
2. Fehler..
  - a) ..lassen sich einsehen,
  - b) ..lassen sich basierend auf ihren Eigenschaften filtern,
  - c) ..lassen sich gruppieren.
  - d) Fehlergruppen lassen sich in Graphen visualisieren (bspw. Histogramm der Häufigkeit).
3. Metriken..
  - a) ..lassen sich in Graphen visualisieren.
4. Traces..
  - a) ..lassen sich einsehen,
  - b) ..lassen sich basierend auf ihren Eigenschaften filtern,
  - c) ..lassen sich als ein Trace-Gantt-Diagramm darstellen.
5. Session-Replay-Daten
  - a) Mithilfe der Session-Replay-Daten soll eine videoähnliche Nachstellung einer Sitzung erstellt werden.

### 4.3.2 Architektur

Auf Basis der zuvor beschriebenen Grunddisziplinen wird nun eine beispielhafte Umsetzung dessen konzipiert. Genauer wird eine Architektur vorgeschlagen, welche auf die zuvor betrachteten Methoden und Praktiken zurückgreift, um eine verbesserter Nachvollziehbarkeit zu erreichen. Speziell wird im Folgeabschnitt zudem vorgeschlagen, welche Werkzeuge oder Technologien zum Einsatz kommen sollen und wie diese Komponenten miteinander kommunizieren.

Die genaue Erhebung der Daten ist Teil der Implementierung und wird hier nicht näher bestimmt. Jedoch ergeben sich aus der zuvor definierten Anforderungen zur Erhebung bereits Datenkategorien, welche von einem entsprechenden Partnersystem zu konsumieren und verarbeiten sind. Es wird zwar nicht auf spezielle Werkzeuge oder Technologien eingegangen, aber es lassen sich bereits Partnersysteme bestimmen, auf Basis der zuvor identifizierten Funktionsbereiche. Hierbei wurde zudem versucht möglichst viele Bereiche über die gleichen Partnersysteme abzubilden (vgl. Anforderung 3100), die Machbarkeit einer solchen Verknüpfung basiert auf den Ergebnissen von Kapitel 3.

So soll für die Verarbeitung von Fehler-, Log-, und Metrikdaten ein einzelnes Partnersystem verantwortlich sein, welches ermöglicht diese zu konsumieren, speichern, durchsuchen und diese zu visualisieren. Grund hierfür ist, dass die Daten gemeinsame Eigenschaften besitzen, die eine gemeinsame Verarbeitung erlauben. In der Abbildung 4.9 ist dieses System als „Log- und Monitoringplattform“ vorzufinden.

Um Traces zu konsumieren und den Betreibern und Entwicklern aufbereitet zu visualisieren, soll ein weiteres Partnersystem eingesetzt werden. Dieses System wurde als notwendig empfunden, da kein Werkzeug identifiziert werden konnte, welches neben Traces noch andere Datenkategorien zufriedenstellend abdecken kann. Auf Basis von OpenTelemetry könnten sich jedoch in Zukunft Technologien entwickeln, welches alle 3 Datenkategorien von OpenTelemetry unterstützt: Metriken, Traces und Logs. Es wurde sich zudem gegen eine weitreichende Monitoringplattform, wie z. B. New Relic oder Dynatrace, entschieden, denn hier wurde identifiziert, dass diese nicht ausreichend flexibel für verschiedene Projekte sind und zudem auch nicht erlauben, dass einzelne Komponenten ausgetauscht oder entfernt werden können.

Es ist zudem ein drittes System notwendig, um die gewünschte Funktionalität des Session-Replays einzubinden. Session-Replay ist ein spezielles und sehr konkretes Aufgabengebiet und es konnte kein Werkzeug identifiziert werden, welches sich nicht nur auf dieses Gebiet spezialisiert.



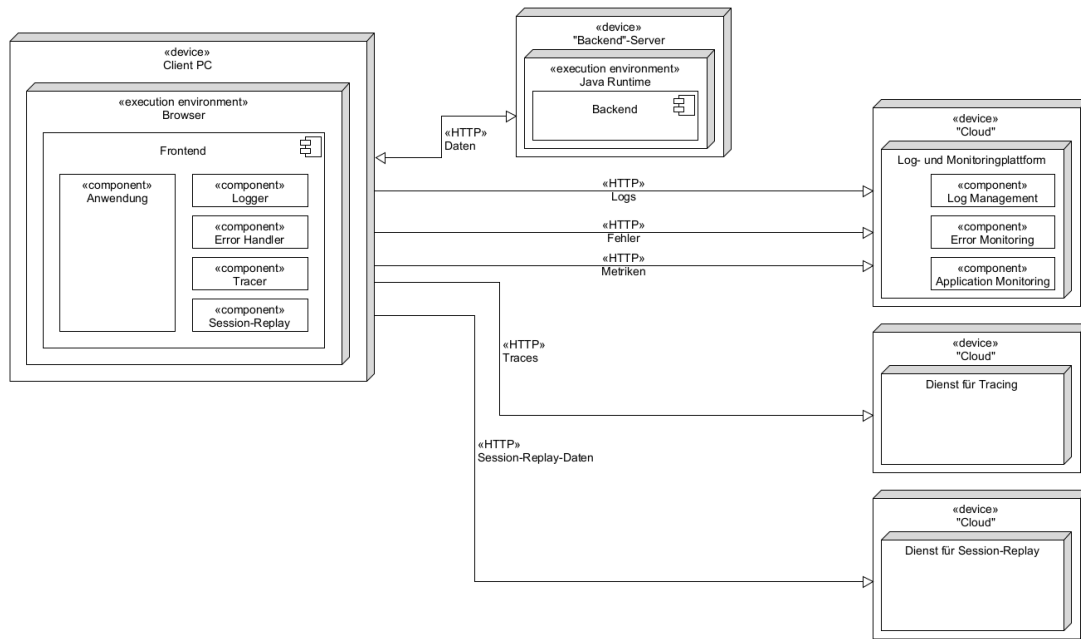


Abb. 4.9: Grobe Architektur

### 4.3.3 Technologie-Stack

Auf Basis der zuvor erstellten Architektur wird sich nun für spezielle Technologien entschieden, mit der diese Architektur umgesetzt werden soll. Weiterhin wird behandelt, wie die Daten vom Frontend aus erhoben werden sollen und wie sie an die Partnersysteme gelangen.

Für die „Log- und Monitoringplattform“ wurde sich für Splunk entschieden, denn auf Basis der Evaluierung konnte festgestellt werden, dass Splunk die drei gewünschten Datenkategorien Logs, Metriken und Fehler zufriedenstellend unterstützt. Es wurde sich gegen New Relic und Dynatrace entschieden, da es diesen Werkzeugen an Flexibilität fehlt und sie viele Funktionen anbieten, die für die Lösung nicht notwendig sind. Eine weitere Alternative ist der Elastic Stack, welcher jedoch nicht näher evaluiert wurde. Somit wird Splunk eingesetzt, aber für eine äquivalente Lösung kann Splunk durch ein gleichwertiges Werkzeug ausgetauscht werden.

Für das Partnersystem, welches sich mit den Tracedaten befasst, wurde sich für Jaeger entschieden. Die Entscheidung wurde auf der Basis getroffen, dass Jaeger einen moderner Tracingdienst darstellt, welcher zudem quelloffen entwickelt wird. Weiterhin ist in Zukunft eine Unterstützung des OpenTelemetry-Standards geplant, was durch die standardisierte Schnittstelle die Anbindung an bestehende Systeme vereinfachen wird. Des

Weiteren erfüllt Jaeger alle aufgestellten Kriterien und erzeugt zudem ein Abbild der Systemarchitektur auf Basis der Traces.

Um die Session-Replay-Funktionalität einzubringen wird in diesem Konzept LogRocket vorgeschlagen. LogRockets Nachstellung einer Sitzung ist nicht nur wie gefordert video-ähnlich, sondern auch interaktiv. Die gesamte HTML-Struktur wird nachgestellt und kann so zu jedem Zeitpunkt begutachtet werden.

Der sich daraus ergebende Technologiestack, angewandt auf die Architektur, ist in Abbildung 4.10 zu betrachten. Wie bei Splunk erwähnt sind auch die anderen Werkzeuge durch gleichwertige Werkzeuge ersetzbar, so können individuelle Anpassungen erfolgen und betriebliche Gegebenheiten zu berücksichtigen.

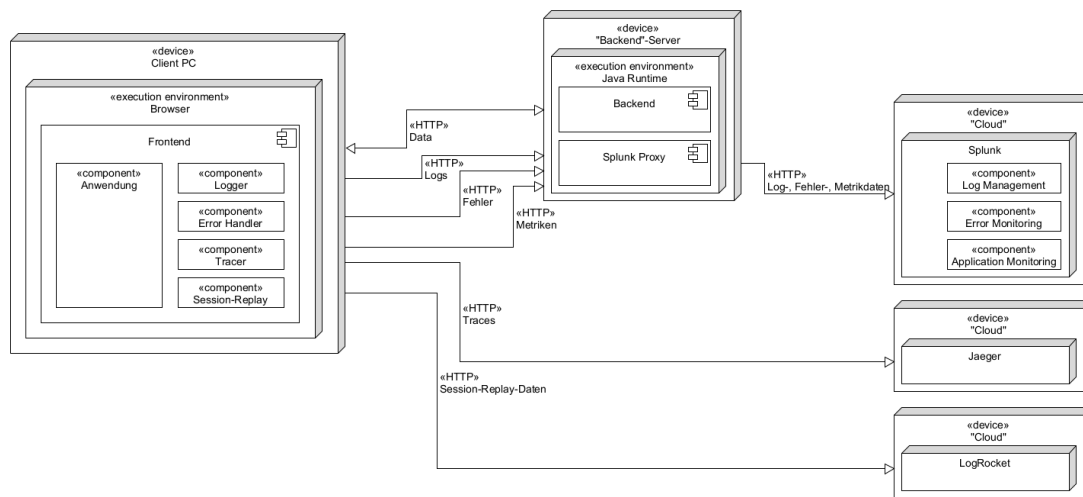


Abb. 4.10: Architektur mit speziellen Technologien

### 4.3.4 Übertragbarkeit

Übertragbarkeit beschäftigt sich mit der Eigenschaft eines Ansatzes in verschiedene Situationen anwendbar zu sein. Ein Ansatz ist nicht übertragbar, wenn zu vielen Annahmen über die Situation getroffen werden.

Das zuvor definierte Konzept wurde getrennt von der Demoanwendung erstellt und ist somit nicht auf dessen Eigenschaften beschränkt. Das Konzept nimmt dennoch an, dass es sich um eine Webanwendung handelt, auf das es anzuwenden ist. Weiterhin wird angenommen, dass Quellcodeänderungen vorgenommen werden können und das Partersysteme hinzugefügt sowie angebunden werden können.

Es wird jedoch nicht angenommen, dass die Partnersysteme exakt von den vorgeschlagenen Technologien realisiert werden. Vielmehr wurde die Funktionsgruppen definiert, zusammengefasst und darauf basierend eine Auswahl getroffen. Sowohl die Zusammenfassung der Funktionsgruppen als auch die Auswahl der eigentlichen Technologien sind individuell änderbar, sodass ein angepasstes aber äquivalent hilfreiches Konzept resultiert.

Somit lässt sich abschließend betrachten, dass das Konzept eine akzeptable Übertragbarkeit aufweist. Eine tiefergehende Nachbetrachtung der Übertragbarkeit erfolgt in Abschnitt 5.3, im Anschluss an die Implementierung. Hierbei kann es zu Abweichungen zu dieser Bewertung der Übertragbarkeit kommen, aufgrund von Implementierungsdetails oder einem geänderten Vorgehen. Auf Basis des beleuchteten Konzeptes, wird im nächsten Abschnitt die eigentliche Umsetzung beschrieben.

## 4.4 Implementierung

### 4.4.1 Backend

Wie in Unterabschnitt 4.2.2 beschrieben, wurden die Dienste mit Eclipse MicroProfile umgesetzt. Neben den standardmäßig enthaltenen Bibliotheken, gibt es hierbei aber auch unterstützte optionale Bibliotheken, wie Implementierungen von **OpenAPI**, **OpenTracing**, **Fault Tolerance** und vieler weiterer [Ecl21].

Um Traces von den Microservices zu sammeln, wurde die OpenTracing Implementierung sowie ein Jaeger-Client [Jae21] zum Exportieren der Daten hinzugezogen. Mit dieser Anbindung lassen sich per Annotation (vgl. Quellcode 4.6) alle zu tracenden Businessmethoden definieren, die dann automatisch getraced und über den Jaeger-Client an Jaeger gesendet werden. Bei jedem Microservice wurde diese Annotation dann an relevante Methoden geschrieben und der Jaeger-Client konfiguriert, was automatisch zu der Übertragung von verteilten Traces in Jaeger führte.

```
1 @ApplicationScoped
2 public class OrderService {
3
4     @Inject
5     private ValidationService validation;
6
7     @Traced(operationName = "OrderService.placeOrder")
8     public Receipt placeOrder(Order order, ShoppingCart shoppingCart) {
9         validation.validateBillingAddress(order.getBillingAddress());
10
11         validation.validateShippingData(order.getShippingData());
12
13         /* calculate Order and prepare Receipt */
```

```

14
15     return new Receipt(/* receipt data */);
16 }
17 }

```

Quellcode 4.6: Beispielhafter Einsatz der @Traced-Annotation

In Jaeger erzeugt der o. g. Quellcode die in Abbildung 4.11 zu sehenden Spans. Neben den Traces werden keine weiteren Daten von Backend-Komponenten erhoben, da das Hauptaugenmerk der Arbeit im Frontend liegt.

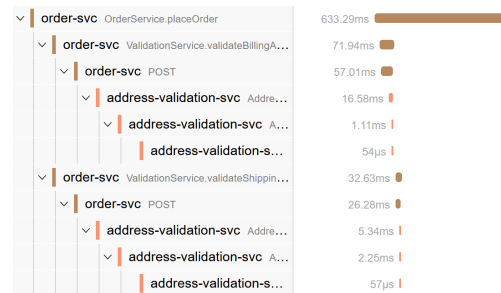


Abb. 4.11: Ausschnitt des Traces zu Quellcode 4.6

### 4.4.2 Frontend

#### 4.4.2.1 Traces und Metriken

Das Frontend erhebt ebenso wie das Backend Traces, aber zusätzlich werden auch Metriken, Logmeldungen und Fehler erhoben und gemeldet. Traces und Metriken werden auf Basis von OpenTelemetry JavaScript Komponenten [Ope21a] erhoben. Genauer werden diese Komponenten in einem Angular Modul (siehe Quellcode 4.7) initialisiert und der restlichen Anwendung über „providers“ zur Verfügung gestellt. Hierbei wird der SPA ein **Tracer** bereitgestellt, mit dem Spans aufgezeichnet werden können, ein **Meter**, welches es erlaubt Metriken zu erstellen, und eine **requestCounter**-Metrik, welches die Aufzeichnung der Aufrufanzahl schnittstellenübergreifend erlaubt.

```

1 import { NGXLogger } from 'ngx-logger';
2
3 import { Tracer } from '@opentelemetry/tracing'
4 import { Meter } from '@opentelemetry/metrics';
5
6 import { AppConfig, APP_CONFIG } from 'src/app/app-config-module';
7 import { SplunkForwardingService } from 'src/app/shared/splunk-forwarding-
   svc/splunk-forwarding.service';
8
9 const tracerFactory = (log: NGXLogger, config: AppConfig): Tracer => {
10   /* Logik zum Erstellen des Tracers */
11   return tracer;
12 };
13
14 const meterFactory = (splunkFwdSvc: SplunkForwardingService): Meter => {
15   /* Logik zum Erstellen des Meters */
16   return meter;
17 };
18

```

```
19 const requestCountMetric = (meter: Meter): CounterMetric => {
20   return meter.createCounter('requestCount') as CounterMetric;
21 };
22
23
24 @NgModule({
25   providers: [
26     {
27       provide: Tracer,
28       useFactory: tracerFactory,
29       deps: [ NGXLogger, APP_CONFIG ]
30     },
31     {
32       provide: Meter,
33       useFactory: meterFactory,
34       deps: [ SplunkForwardingService ]
35     },
36     {
37       provide: CounterMetric,
38       useFactory: requestCountMetric,
39       deps: [ Meter ]
40     },
41   ]
42 })
43 export class AppObservabilityModule {
44   constructor() {}
45 }
```

Quellcode 4.7: Quellcode des Moduls „app-observability.module.ts“

Im Quellcode 4.8 ist die Benutzung des zur Verfügung gestellten **Tracers** zu sehen, hierbei wird ein Span erstellt und bei Schnittstellenaufrufen an die jeweiligen Services übergeben.

```
1 @Injectable({ providedIn: 'root' })
2 export class ShoppingCartDataSource extends DataSource<ShoppingCartItem> {
3   constructor(
4     private log: NGXLogger,
5     private cartService: ShoppingCartService,
6     private localizationService: LocalizationService,
7     private tracer: Tracer
8   ) {
9     super();
10  }
11
12  getAndMapShoppingCart(shoppingCartId: string) {
13    const span = this.tracer.startSpan( 'ShoppingCartDataSource.
14      getAndMapShoppingCart', { /* Attribute */ } );
15
16    this.log.debug('getAndMapShoppingCart(): requesting translations');
17    const translations$ = this.localizationService.getTranslations(span);
```

```

17
18     this.log.debug('getAndMapShoppingCart(): requesting shoppingCart');
19     const shoppingCart$ = this.cartService.getShoppingCart(shoppingCartId,
        span);
20
21     return /* ... */;
22 }
23 }

```

Quellcode 4.8: Datenquelle zum Abrufen und Zusammenführen der Artikeldaten

Beispielhaft im Dienst zum Abrufen der Übersetzungsdaten (vgl. Quellcode 4.9) wird der übergebene Span als Elternspan benutzt. Bei dem eigentlichen HTTP-Aufruf wird zudem ein HTTP-Header **uber-trace-id** angereichert, den der dort laufende Jaeger-Client interpretiert [Jae21] und daraus die Beziehung zu den Frontend-Spans herstellt. Zusätzlich zum Tracing wird hierbei auch die Metrik **requestCounter** inkrementiert.

```

1 @Injectable({ providedIn: 'root' })
2 export class LocalizationService {
3     constructor(
4         private log: NGXLogger,
5         private http: HttpClient,
6         private tracer: Tracer,
7         private traceUtil: TraceUtilService,
8         private requestCounter: CounterMetric
9     ) {}
10
11     public getTranslations(parentSpan?: api.Span) {
12         this.log.info('getTranslations(): requesting translations');
13
14         // start span with provided span as a parent
15         const span = this.tracer.startSpan(
16             'LocalizationService.getTranslations',
17             { /* attributes */},
18             parentSpan && api.setSpan(api.context.active(), parentSpan)
19         );
20
21         // generate a jaeger-compatible trace header from OTel span
22         const jaegerTraceHeader = this.traceUtil.
            serializeSpanContextToJaegerHeader(span.context());
23
24         // increment the requestCounter metric
25         this.requestCounter.add(1, { 'component': 'LocalizationService' });
26
27         return this.http.get<Localization>(
28             this.localizationServiceUrl,
29             { headers: { 'uber-trace-id': jaegerTraceHeader } }
30         )
31         .pipe(
32             tap(

```

```

33         (val) => {
34             this.log.info('getTranslations(): returnVal = ', val);
35
36             span.end();
37         },
38         (err) => {
39             // handle and record exception
40             span.recordException({ code: err.status, name: err.name,
message: err.message });
41             span.end();
42         }
43     ),
44 );
45 }
46 }

```

Quellcode 4.9: Service zum Abrufen der Übersetzungsdaten

Es wurde sich für die OpenTelemetry Implementierung für Tracing und Metriken im Frontend entschieden, da wie in Unterabschnitt 3.2.7 beschrieben, OpenTelemetry einen vielversprechenden Standard darstellt. Weiterhin konnte keine Bibliothek identifiziert werden, die die Traces erhebt und direkt nach Jaeger sendet. Es gibt zwar einen Jaeger-Client für Node.js<sup>1</sup>, jedoch befindet sich das browserkompatible Pendant<sup>2</sup> seit 2017 in den Startlöchern. Weiterhin existiert ein OTEL Exporter für Jaeger<sup>3</sup>, welcher jedoch auch nur mit Node.js funktioniert.

Die gesammelten OTEL Tracingdaten werden über einen Standard-Exporter an das „Backend4Frontend“ gesendet, welcher diese dann in ein Jaeger-konformes Format umwandelt und sie dann subsequent an Jaeger überträgt. Die Metrikdaten werden jedoch bereits im Frontend konvertiert, in ein Splunk-kompatibles Logformat. Nach der Konvertierung werden die Daten an den **SplunkForwardingService** übergeben, welcher im folgenden Abschnitt näher beschrieben wird.

### 4.4.2.2 Logging

Das Logging im Frontend wurde über das npm [npm21] Paket **ngx-logger**<sup>4</sup> realisiert, welches eine speziell an Angular angepasste Logging-Lösung darstellt. Da dieses Pakete extra an Angular angepasst ist, lässt es sich ohne großen Aufwand als Modul einbinden, vgl. Quellcode 4.10.

<sup>1</sup>Jaeger-Client für Node.js: <https://github.com/jaegertracing/jaeger-client-node>

<sup>2</sup>Jaeger-Client für Browser: <https://github.com/jaegertracing/jaeger-client-javascript/>

<sup>3</sup>OTel Jaeger Exporter: <https://github.com/open-telemetry/opentelemetry-js/tree/main/packages/opentelemetry-exporter-jaeger>

<sup>4</sup>ngx-logger auf GitHub: <https://github.com/dbfannin/ngx-logger>

```

1 import { LoggerModule, NgxLoggerLevel } from 'ngx-logger';
2
3 @NgModule({
4   declarations: [
5     /* Komponenten */
6   ],
7   imports: [
8     /* andere Module */
9     LoggerModule.forRoot({
10       level: NgxLoggerLevel.DEBUG,
11     }),
12   ],
13 })
14 export class AppModule { }

```

Quellcode 4.10: Ausschnitt des Hauptmoduls `app.module.ts`

Wie in den vorherigen Codebeispielen zum Tracing zu sehen war, kann ein `NGXLogger` im Konstruktor von Komponenten und Diensten injected werden. Logmeldungen die hiermit erfasst werden, werden je nach Konfiguration und Loglevel der jeweiligen Meldung in die Browserkonsole geschrieben. Über einen `NGXLoggerMonitor` lassen sich die Logmeldungen anzapfen, wie in Quellcode 4.11 zu sehen ist. Hierbei werden die Logmeldungen in ein Splunkformat übertragen und dann über den `SplunkForwardingService` an das „Backend4Frontend“ übertragen. Eine direkte Übertragung an Splunk ist nicht möglich, da Splunk nicht mit kompatiblen CORS-Headern antwortet. Das Backend4Frontend reicht neben dem Weiterleiten auch die Meldungen mit Kontextinformationen, wie der User-IP, der Browserversion usw. an.

```

1 @Injectable({ providedIn: 'root' })
2 export class SplunkLoggingMonitor extends NGXLoggerMonitor {
3   constructor(
4     private log: NGXLogger,
5     private splunk: SplunkForwardingService
6   ) {
7     super();
8   }
9
10  onLog(logObject: NGXLogInterface): void {
11    const params = this.makeParamsPrintable(logObject.additional);
12
13    const logEvent = {
14      sourcetype: 'log',
15      event: {
16        severity: logObject.level,
17        message: logObject.message,
18        ...params,
19        fileName: logObject.fileName,
20        lineNumber: logObject.lineNumber,
21        timestamp: logObject.timestamp

```



```

22         }
23     };
24
25     this.splunk.forwardEvent(logEvent);
26 }
27 }

```

Quellcode 4.11: Implementierung des NGXLoggerMonitor-Interfaces

#### 4.4.2.3 Fehler

Die ErrorHandler-Hook von Angular übermittelt aufgetretene und unbehandelte Fehler an den `SplunkForwardingErrorHandler`. Weiterhin ist der ErrorHandler `Injectable` in andere SPA Klassen, wo er bspw. bei den Schnittstellen dazu benutzt wird, dass auch behandelte Fehler an Splunk zu übermitteln.

Wird ein Fehler gemeldet, werden zunächst die Fehlerinformationen in einen Splunkdatensatz konvertiert und dann über den zuvor behandelten `SplunkForwardingService` an Splunk weitergeleitet. Neben diesem Verhalten wird zusätzlich auch der Fehler an LogRocket übermittelt, damit dieser im Video des Session-Replays gesondert angezeigt wird.

```

1  @Injectable({ providedIn: 'root' })
2  export class SplunkForwardingErrorHandler extends ErrorHandler {
3      private splunkForwarding: SplunkForwardingService;
4
5      constructor(injector: Injector) {
6          super();
7
8          this.splunkForwarding = injector.get(SplunkForwardingService);
9      }
10
11     handleError(error, optionalData?: any): void {
12         LogRocket.captureException(error);
13
14         const entry: SplunkEntry = {
15             sourcetype: 'error',
16             event: {
17                 ...optionalData,
18                 frontendModel: window.frontendModel,
19                 // see https://developer.mozilla.org/en-US/docs/Web/
                JavaScript/Reference/Global_Objects/Error
20                 name: error.name,
21                 message: error.message,
22                 stack: error.stack,
23                 fileName: error.fileName, // non-standard
24                 lineNumber: error.lineNumber, // non-standard
25                 columnNumber: error.columnNumber // non-standard

```

```

26         }
27     };
28
29     this.splunkForwarding.forwardEvent(entry);
30 }
31 }

```

Quellcode 4.12: ErrorHandler zum Abfangen und Weiterleiten von aufgetretenen Fehlern

### 4.4.2.4 Session-Replay

LogRocket wird standardmäßig nicht aktiv, außer wenn der Nutzer explizit der Aufzeichnung zustimmt [cite]. Dies bedeutet jedoch auch, dass bis zur Zustimmung keine Sitzungsdaten aufgenommen wurden. Wenn der Nutzer zustimmt, wird, wie im Quellcode 4.13 zu sehen ist, LogRocket initialisiert und mit identifizierenden Daten angereichert. Nach dem Warenkorbdialog und nach der Eingabe der Rechnungsadresse werden zudem weitere identifizierende Daten an LogRocket übermittelt. Die Aufnahme der Sitzung läuft größtenteils autonom, lediglich behandelte Fehler müssen LogRocket manuell übermittelt werden.

```

1  @Component({
2    selector: 'app-checkout',
3    templateUrl: './checkout.component.html',
4    styleUrls: ['./checkout.component.css']
5  })
6  export class CheckoutComponent implements OnInit {
7    /* andere Methoden */
8
9    activateLogRocket() {
10      // initialize session recording
11      LogRocket.init('<app-id>', {});
12      // start a new session
13      LogRocket.startNewSession();
14      // make sessionURL accessible
15      LogRocket.getSessionURL((sessionURL) => {
16        window.logrocketData.sessionURL = sessionURL;
17      });
18      // pass unique "session"-id to LogRocket
19      LogRocket.identify(window.customer.shoppingCartId);
20    }
21
22    onShoppingCartSubmit(data: CheckoutShoppingCartInfo): void {
23      // enrich LogRocket data with ShoppingCart data
24      LogRocket.identify(window.customer.shoppingCartId, {
25        itemCount: data.itemCount,
26        totalSum: data.totalSum
27      });
28    }

```

```
29 |
30 | onBillingAddressSubmit(data: CheckoutBillingAddress): void {
31 |     // enrich LogRocket data with BillingAddress data
32 |     LogRocket.identify(window.customer.shoppingCartId, {
33 |         name: data.firstName + ' ' + data.lastName,
34 |         email: data.email
35 |     });
36 | }
37 | }
```

Quellcode 4.13: Initialisierung von LogRocket in der Hauptkomponente

## 5 Ergebnis

### 5.1 Demonstration

*Nach Abschluss der Implementierung, soll die Erweiterung auf nicht-technische Weise veranschaulicht werden. Hier soll dargestellt werden, wie die Nachvollziehbarkeit nun verbessert worden ist.*

### 5.2 Kriterien

*Die zuvor definierten Kriterien in 4.1 sollen hier überprüft werden.*

*Auch interessant: Wie sieht der Datendurchsatz generell aus? Wie sieht der Datendurchsatz pro Komponente aus?*

### 5.3 Übertragbarkeit

*Wie gut lassen sich die ermittelten Ergebnisse im PoC auf andere Projekte im selben Umfeld übertragen?*

### 5.4 Einschätzung von anderen Entwicklern (optional)

*Dieser Abschnitt kann ggf. wegfallen, wenn nicht genügend Zeit besteht oder der Nutzen nicht den Aufwand gerechtfertigt.*

*In diesem Abschnitt werden Frontend-Entwickler mit der Demo-Anwendung konfrontiert, einerseits mit und andererseits ohne die Lösung. Daraufhin werden den Entwicklern bspw. folgende Fragen gestellt:*

- 1. Wie gut entspricht die Demo-Anwendung einem realen Szenario?*
- 2. Sind die vorgestellten Probleme realitätsnah?*

3. *Wie gut lassen sich die Probleme ohne die Lösung beheben?*
4. *Wie gut lassen sich die Probleme mit der Lösung beheben?*
5. *Ist der Lösungsansatz zu komplex?*
6. *Gibt es Bedenken zum Lösungsansatz?*

## 6 Abschluss

### 6.1 Fazit

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.*

### 6.2 Ausblick

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.*

## 7 Anhang

### 7.1 Studien zur Browserkompatibilität

Im Unterabschnitt 2.1.1 wurde die Anzahl an Studien zur Browserkompatibilität dargestellt. Die Daten hierfür wurden über die Literatursuchmaschine „Google Scholar“ am 14.01.2021 abgerufen

Für die Suche wurde folgender Suchterm benutzt:

`"cross browser" compatibility|incompatibility|inconsistency|XBI`

Die Trefferanzahl für ein spezielles Jahr wurde jeweils als Datenpunkt benutzt. Dies soll dazu dienen, um einen ungefähren Trend der Literatur zu erkennen.

Jahr	Treffer
2015	299
2016	246
2017	286
2018	238
2019	187
2020	160

Tab. 7.1: Suchtreffer zu Studien über Browserkompatibilität

# Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig angefertigt und mich keiner fremden Hilfe bedient sowie keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäß veröffentlichten oder nicht veröffentlichten Schriften und anderen Quellen entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Dortmund, am .....  
(Unterschrift)



# Abkürzungs- und Erklärungsverzeichnis

Ajax Asynchronous JavaScript and XML

CDN Content Delivery Network

Clientseitiges Rendering Der Server stellt dem Client lediglich die Logik und die notwendigen Daten bereit, die eigentliche Inhaltsgenerierung geschieht im Client. Beispiel siehe Abschnitt 2.2

CNCF Cloud Native Computing Foundation

CORS Cross-Origin Resource Sharing

CPU Central Processing Unit, auf Deutsch „Prozessor“.

CSP Content-Security-Policy

HTTP Hyper-Text-Transfer-Protocol

OTel OpenTelemetry

PoC Proof-of-Concept

SaaS Software-as-a-Service

Serverseitiges Rendering Die darzustellenden Inhalte, werden beim Server generiert und der Client stellt diese dar. Beispielsweise sind Anwendungen mit PHP oder auch eine Java Web Application

UI User-Interface

W3C World Wide Web Consortium

XBI Cross-Browser-Incompatibilities

XHR XMLHttpRequest

XSS Cross-Site-Scripting

# Abbildungsverzeichnis

2.1	Studien zur Browserkompatibilität, eigene Darstellung (vgl. 7.1)	4
2.2	Flowchart über den Ablauf von Ajax-Anfragen mit CORS [Blu15]	8
3.1	Kausale Beziehung zwischen Spans. Eigene Darstellung.	12
3.2	Zeitliche Beziehung zwischen Spans. Eigene Darstellung.	12
3.3	Fehlerbericht in der Instagram App [Fac20a]	12
3.4	Mitschneiden von DOM-Events, Abb. aus [BBKE13]	14
3.5	Abspielen von DOM-Events, Abb. aus [BBKE13]	15
3.6	Schaubild einer Lösung auf Basis von OTel [Ope20c]	16
3.7	OTel Komponenten [Dyn20c]	17
3.8	Ausschnitt eines Session Replays bei LogRocket	21
3.9	Trace-Detailansicht. Quelle: Don Schenck [Sch20]	22
3.10	Dienst-Abhängigkeits-Graph. Quelle: Yuri Shkuro [Shk20]	23
3.11	Zoom von Abbildung 3.12, Abbildung aus [ASD <sup>+</sup> 20]	23
3.12	TraVistas Gantt-Diagramm, Abbildung aus [ASD <sup>+</sup> 20]	24
4.1	Demoanwendung: Deployment-Diagramm, Quelle: Eigene Darstellung	38
4.2	Demoanwendung: Startseite „Warenkorb“	39
4.3	Demoanwendung: Seite „Rechnungsadresse“	40
4.4	Demoanwendung: Seite „Lieferdaten“	41
4.5	Demoanwendung: Seite „Zahlungsdaten“	42
4.6	Demoanwendung: Seite „Bestellübersicht“	43
4.7	Demoanwendung: Finale Seite „Bestellbestätigung“	44
4.8	Fehlende Texte	45
4.9	Grobe Architektur	50
4.10	Architektur mit speziellen Technologien	51
4.11	Ausschnitt des Traces zu Quellcode 4.6	53

## Tabellenverzeichnis

4.1	Merkmale nach dem Kano-Modell der Kundenzufriedenheit . . . . .	26
4.2	Kategorien der Anforderungen . . . . .	27
4.3	Quellen der Anforderungen . . . . .	27
4.4	Beispiel einer Anforderung . . . . .	28
7.1	Suchtreffer zu Studien über Browserkompatibilität . . . . .	64

## Quellcodeverzeichnis

4.1	Demoanwendung: Gherkin Definition zum Feature „Warenkorb“ . . . . .	33
4.2	Demoanwendung: Gherkin Definition zum Feature „Rechnungsadresse“ . .	34
4.3	Demoanwendung: Gherkin Definition zum Feature „Lieferadresse“ . . . . .	35
4.4	Demoanwendung: Gherkin Definition zum Feature „Zahlungsdaten“ . . . . .	36
4.5	Demoanwendung: Gherkin Definition zum Feature „Bestellung abschließen“	36
4.6	Beispielhafter Einsatz der @Traced-Annotation . . . . .	52
4.7	Quellcode des Moduls „app-observability.module.ts“ . . . . .	53
4.8	Datenquelle zum Abrufen und Zusammenführen der Artikeldaten . . . . .	54
4.9	Service zum Abrufen der Übersetzungsdaten . . . . .	55
4.10	Ausschnitt des Hauptmoduls <code>app.module.ts</code> . . . . .	57
4.11	Implementierung des <code>NGXLoggerMonitor</code> -Interfaces . . . . .	57
4.12	ErrorHandler zum Abfangen und Weiterleiten von aufgetretenen Fehlern .	58
4.13	Initialisierung von LogRocket in der Hauptkomponente . . . . .	59

# Literaturverzeichnis

- [ABC<sup>+</sup>16] AHMED, Tarek M. ; BEZEMER, Cor-Paul ; CHEN, Tse-Hsun ; HASSAN, Ahmed E. ; SHANG, Weiyi: Studying the effectiveness of application performance management (APM) tools for detecting performance regressions for web applications: an experience report. In: *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)* IEEE, 2016, S. 1–12
- [AMWR20] ASROHAH, Hanun ; MILAD, Mohammad K. ; WIBOWO, Achmad T. ; RHOFITA, Erry I.: Improvement of academic services using mobile technology based on single page application. In: *Telfor Journal* 12 (2020), Nr. 1, S. 62–66
- [ASD<sup>+</sup>20] ANAND, Vaastav ; STOLET, Matheus ; DAVIDSON, Thomas ; BESCHASTNIKH, Ivan ; MUNZNER, Tamara ; MACE, Jonathan: Aggregate-Driven Trace Visualizations for Performance Debugging. In: *arXiv preprint arXiv:2010.13681* (2020)
- [BBKE13] BURG, Brian ; BAILEY, Richard ; KO, Andrew J. ; ERNST, Michael D.: Interactive Record/Replay for Web Application Debugging. In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*, 2013, S. 473–484
- [BJS<sup>+</sup>08] BETTENBURG, Nicolas ; JUST, Sascha ; SCHRÖTER, Adrian ; WEISS, Cathrin ; PREMRAJ, Rahul ; ZIMMERMANN, Thomas: What makes a good bug report? In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008, S. 308–318
- [Blu15] BLUESMOON: *Flowchart showing Simple and Preflight XHR.svg*. [https://commons.wikimedia.org/wiki/File:Flowchart\\_showing\\_Simple\\_and\\_Preflight\\_XHR.svg](https://commons.wikimedia.org/wiki/File:Flowchart_showing_Simple_and_Preflight_XHR.svg), 2015. – [Online; abgerufen am 09.11.2020]
- [Bra16] BRAUN, Michael: Nicht-funktionale Anforderungen. In: *Juristisches IT-Projektmanagement Lehrstuhl für Programmierung und Softwaretechnik Ludwig-Maximilians-Universität München* (2016). – S. 3–5
- [BT19] BRUHIN, Florian ; TAVERNINI, Luca: *Crashbin – Dokumentation*. Hochschule für Technik Rapperswil & Fachhochschule Ostschweiz, 2019

- [CGL<sup>+</sup>15] CITO, Jürgen ; GOTOWKA, Devan ; LEITNER, Philipp ; PELETTE, Ryan ; SULJOTI, Dritan ; DUSTDAR, Schahram: Identifying Web Performance Degradations through Synthetic and Real-User Monitoring. In: *J. Web Eng.* 14 (2015), Nr. 5&6, S. 414–442
- [CPO14] CHOUDHARY, Shaunik R. ; PRASAD, Mukul R. ; ORSO, Alessandro: X-PERT: a web application testing tool for cross-browser inconsistency detection. In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, S. 417–420
- [Doc20] DOCKER: *Docker overview / Docker Documentation*. <https://docs.docker.com/get-started/overview/>, 2020. – [Online; abgerufen am 15.02.2021]
- [Dyn20a] DYNATRACE: *Dynatrace joins the OpenTelemetry project*. <https://www.dynatrace.com/news/blog/dynatrace-joins-the-opentelemetry-project/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Dyn20b] DYNATRACE: *The Leader in Cloud Monitoring / Dynatrace*. <https://www.dynatrace.com/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Dyn20c] DYNATRACE: *What is OpenTelemetry? Everything you wanted to know*. <https://www.dynatrace.com/news/blog/what-is-opentelemetry/>, 2020. – [Online; abgerufen am 20.11.2020]
- [EAN17] ENGLEHARDT, Steven ; ACAR, Gunes ; NARAYANAN, Arvind: No boundaries: Exfiltration of personal data by session-replay scripts. In: *Freedom to tinker* 15 (2017), S. 3–4
- [Ec121] ECLIPSE FOUNDATION: *MicroProfile / projects.eclipse.org*. <https://projects.eclipse.org/projects/technology.microprofile>, 2021. – [Online; abgerufen am 10.02.2021]
- [Fac20a] FACEBOOK: *Instagram App Screenshot*. <https://www.instagram.com/>, 2020
- [Fac20b] FACEBOOK: *React - A JavaScript library for building user interfaces*. <https://reactjs.org>, 2020. – [Online; abgerufen am 12.10.2020]
- [Fil20] FILIPE, Ricardo Ângelo S.: *Client-Side Monitoring of Distributed Systems*, Universidade de Coimbra, Diss., 2020
- [Fra20] FRANEY, Logan: *What is observability? / Dynatrace blog*. <https://www.dynatrace.com/news/blog/what-is-observability/>, 2020
- [Fre91] FREEDMAN, Roy S.: Testability of software components. In: *IEEE transactions on Software Engineering* 17 (1991), Nr. 6, S. 553–564

- [Fun20] FUNCTIONAL SOFTWARE: *About Sentry / Sentry*. <https://sentry.io/about/>, 2020. – [Online; abgerufen am 23.11.2020]
- [GB20] GREIF, Sacha ; BENITTE, Raphaël: *State of JS 2020: Front-end Frameworks*. <https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>, 2020. – [Online; abgerufen am 15.11.2020]
- [Goo20] GOOGLE: *Angular*. <https://angular.io>, 2020. – [Online; abgerufen am 12.10.2020]
- [Hop06] HOPMANN, Alex: *The story of XMLHTTP*. <https://web.archive.org/web/20070623125327/http://www.alexhopmann.com/xmlhttp.htm>, 2006. – [Online; abgerufen am 27.10.2020]
- [Hou20] HOUND TECHNOLOGY: *Why Honeycomb - Honeycomb*. <https://www.honeycomb.io/why-honeycomb/>, 2020. – [Online; abgerufen am 23.11.2020]
- [Jae20] JAEGER AUTHORS: *Jaeger: open source, end-to-end distributed tracing*. <https://www.jaegertracing.io/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Jae21] JAEGER AUTHORS: *Client Libraries - Jaeger documentation*. <https://www.jaegertracing.io/docs/1.21/client-libraries/>, 2021. – [Online; abgerufen am 15.02.2021]
- [Jos19] JOSEPHSEN, Dave: iVoyeur: Distributive Tracing. In: *login*: 44 (2019), Nr. 4, S. 56. – ISSN 1044-6397
- [JSD15] JADHAV, Madhuri A. ; SAWANT, Balkrishna R. ; DESHMUKH, Anushree.: Single Page Application using AngularJS. In: *International Journal of Computer Science and Information Technologies* 6 (2015), Nr. 3, S. 2876–2879
- [Ká60] KÁLMÁN, Rudolf E.: On the general theory of control systems. In: *Proceedings First International Conference on Automatic Control, Moscow, USSR*, 1960, S. 481–492
- [Kan68] KANO, Noriaki: Concept of TQC and its Introduction. In: *Kuei* 35 (1968), Nr. 4, S. 20–29
- [Kie10] KIENLE, Holger M.: It's about time to take JavaScript (more) seriously. In: *IEEE software* 27 (2010), Nr. 3, S. 60–62
- [Kop14] *Kapitel Appendix B - History of Web Programming*. In: KOPEC, David: *Evolution of the Web Browser*. Springer, 2014, S. 283–286
- [LGB19] LEGEZA, Vladimir ; GOLUBTSOV, Anton ; BEYER, Betsy: Structured Logging: Crafting Useful Message Content. In: *login*; Summer 2019, Vol. 44 (2019), Nr. 2

- [Log20] LOGROCKET: *LogRocket / Logging and Session Replay for JavaScript Apps*. <https://logrocket.com/>, 2020. – [Online; abgerufen am 23.11.2020]
- [Maj18] MAJORS, Charity: Observability for Emerging Infra: What Got You Here Won't Get You There. In: *SREcon18 Europe/Middle East/Africa (SREcon18 Europe)*. Dusseldorf : USENIX Association, August 2018. – Folien 14–17
- [Mic20a] MICROSOFT: *Microsoft 365 apps say farewell to Internet Explorer 11 and Windows 10 sunsets Microsoft Edge Legacy*. <https://techcommunity.microsoft.com/t5/microsoft-365-blog/microsoft-365-apps-say-farewell-to-internet-explorer-11-and/ba-p/1591666>, 2020. – [Online; abgerufen am 29.10.2020]
- [Mic20b] MICROSOFT: *New year, new browser – The new Microsoft Edge is out of preview and now available for download*. <https://blogs.windows.com/windowsexperience/2020/01/15/new-year-new-browser-the-new-microsoft-edge-is-out-of-preview-and-now-available-for-01.15>, 2020. – [Online; abgerufen am 29.10.2020]
- [MM20a] MOZILLA ; MITWIRKENDE individuelle: *console - Web APIs / MDN*. <https://developer.mozilla.org/en-US/docs/Web/API/Console>, 2020. – [Online; abgerufen am 19.10.2020]
- [MM20b] MOZILLA ; MITWIRKENDE individuelle: *Content Security Policy (CSP) - HTTP / MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP/>, 2020. – [Online; abgerufen am 15.10.2020]
- [MM20c] MOZILLA ; MITWIRKENDE individuelle: *Cross-Origin Resource Sharing (CORS) - HTTP / MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>, 2020. – [Online; abgerufen am 15.10.2020]
- [MM21] MOZILLA ; MITWIRKENDE individuelle: *Ajax - Developer guides / MDN*. <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>, 2021. – [Online; abgerufen am 11.02.2021]
- [New20a] NEW RELIC: *Announcing OpenTelemetry Beta support in New Relic One - New Relic Blog*. <https://blog.newrelic.com/product-news/opentelemetry-beta-support-new-relic-one/>, 2020. – [Online; abgerufen am 20.11.2020]
- [New20b] NEW RELIC: *New Relic / Deliver more perfect software*. <https://opentelemetry.io/registry/>, 2020. – [Online; abgerufen am 20.11.2020]
- [NMMA16] *Kapitel The Microservices Way*. In: NADAREISHVILI, Irakli ; MITRA, Ronnie ; McLARTY, Matt ; AMUNDSEN, Mike: *Microservice architecture: aligning principles, practices, and culture*. O'Reilly Media, Inc., 2016, S. 3–8

- [Nor06] NORTH, Dan: *Introducing BDD*. <https://dannorth.net/introducing-bdd/>, 2006. – [Online; abgerufen am 22.01.2021]
- [npm21] NPM: *npm / About*. <https://www.npmjs.com/>, 2021. – [Online; abgerufen am 15.02.2021]
- [OHH<sup>+</sup>16] OKANOVIĆ, Dušan ; HOORN, André van ; HEGER, Christoph ; WERT, Alexander ; SIEGL, Stefan: Towards performance tooling interoperability: An open format for representing execution traces. In: *European Workshop on Performance Engineering* Springer, 2016, S. 94–108
- [OKSK15] OREN, Yossef ; KEMERLIS, Vasileios P. ; SETHUMADHAVAN, Simha ; KEROMYTIS, Angelos D.: The spy in the sandbox: Practical cache attacks in javascript and their implications. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, S. 1406–1418
- [OPBW06] OSHRY, Matt ; PORTER, Brad ; BODELL, Michael ; W3C, World Wide Web C.: *Authorizing Read Access to XML Content Using the <?access-control?> Processing Instruction 1.0*. <https://www.w3.org/TR/2006/WD-access-control-20060517/>, 2006. – [Online; abgerufen am 27.10.2020]
- [Ope20a] OPENCENSUS: *OpenCensus*. <https://opencensus.io/introduction/#overview>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope20b] OPENTELEMETRY: *About / OpenTelemetry*. <https://opentelemetry.io/about/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope20c] OPENTELEMETRY: *opentelemetry-specification/unified-collection.png*. <https://github.com/open-telemetry/opentelemetry-specification/blob/8e7b2cc17f2c572282c4e5e4d3cc54401749d8ff/specification/logs/img/unified-collection.png>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope20d] OPENTELEMETRY: *Registry / OpenTelemetry*. <https://opentelemetry.io/registry/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Ope20e] OPENTRACING: *The OpenTracing Semantic Specification*. <https://github.com/opentracing/specification/blob/c064a86b69b9d170ace3f4be7dbacf47953f9604/specification.md>, 2020. – [Online; abgerufen am 11.12.2020]
- [Ope20f] OPENTRACING: *What is Distributed Tracing?* <https://opentracing.io/docs/overview/what-is-tracing/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope21a] OPENTELEMETRY: *open-telemetry/opentelemetry-js: OpenTelemetry JavaScript API and SDK*. <https://github.com/open-telemetry/opentelemetry-js>, 2021. – [Online; abgerufen am 15.02.2021]



- [Ope21b] OPENTELEMETRY: *Tracking OTel Spec Issues for GA*. <https://github.com/open-telemetry/opentelemetry-specification/issues/1118>, 2021. – [Online; abgerufen am 29.01.2021]
- [Ope21c] OPENTELEMETRY: *Write guidelines and specification for logging libraries to support OpenTelemetry-compliant logs*. <https://github.com/open-telemetry/opentelemetry-specification/issues/894>, 2021. – [Online; abgerufen am 29.01.2021]
- [Ora20] ORACLE: *Java Debug Wire Protocol*. <https://download.java.net/java/GA/jdk14/docs/specs/jdwp/jdwp-spec.html>, 2020. – [Online; abgerufen am 23.10.2020]
- [OS<sup>+</sup>18] ORIOL, Marc ; STADE, Melanie ; ; FOTROUSI, Farnaz ; NADAL, Sergi ; VARGA, Jovan ; SEYFF, Norbert ; ABELLO, Alberto ; FRANCH, Xavier ; MARCO, Jordi ; SCHMIDT, Oleg: FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring. In: *2018 IEEE 26th International Requirements Engineering Conference (RE)* IEEE, 2018, S. 217–227
- [Pow06] POWERS, Shelley: *Learning JavaScript*. O'Reilly Media Inc., 2006 (Java Series). – ISBN 9780596527464
- [Pro20] PROMETHEUS AUTOREN: *Prometheus - Monitoring system & time series database*. <https://prometheus.io/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ran09] RANGANATHAN, Arun: *cross-site xmlhttprequest with CORS*. <https://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/>, 2009. – [Online; abgerufen am 27.10.2020]
- [Rel21] RELIC, New: *Why the Future Is Open, Connected, and Programmable*. <https://newrelic.com/resources/ebooks/what-is-observability>, 2021
- [Sch20] SCHENCK, Don: *Istio Tracing & Monitoring: Where Are You and How Fast Are You Going?* <https://developers.redhat.com/blog/2018/04/03/istio-tracing-monitoring/>, 2020. – [Online; abgerufen am 17.12.2020]
- [Sen20a] SENTRY: *getsentry/sentry-javascript: Official Sentry SDKs for Javascript*. <https://github.com/getsentry/sentry-javascript>, 2020. – [Online; abgerufen am 23.11.2020]
- [Sen20b] SENTRY: *Self-Hosted Sentry | Sentry Developer Documentation*. <https://develop.sentry.dev/self-hosted/>, 2020. – [Online; abgerufen am 23.11.2020]
- [Shk20] SHKURO, Yuri: *Take OpenTracing for a HotROD ride*. <https://medium.com/opentracing/take-opentracing-for-a-hotrod-ride-f6e3141f7941>, 2020. – [Online; abgerufen am 17.12.2020]

- [Sma19] SMARTBEAR SOFTWARE: *Gherkin Syntax*. <https://cucumber.io/docs/gherkin/>, 2019. – [Online; abgerufen am 14.11.2020]
- [Spl20] SPLUNK: *The Data-to-Everything Platform Built for the Cloud | Splunk*. <https://www.splunk.com/>, 2020. – [Online; abgerufen am 24.11.2020]
- [Sta21] STATCOUNTER: *Desktop Browser Market Share Worldwide (Jan - Dec 2020)*. <https://gs.statcounter.com/browser-market-share/desktop/worldwide/2020>, 01 2021. – [Online; abgerufen am 15.01.2021]
- [STM+20] SCROCCA, Mario ; TOMMASINI, Riccardo ; MARGARA, Alessandro ; VALLE, Emanuele D. ; SAKR, Sherif: The Kaiju Project: Enabling Event-Driven Observability. In: *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems* ACM, 2020, S. 85–96
- [The20] THE LINUX FOUNDATION: *What is Kubernetes? | Kubernetes*. <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>, 2020. – [Online; abgerufen am 15.02.2021]
- [TVNP13] TOVARNÁK, Daniel ; VAEKOVÁ, Andrea ; NOVÁK, Svatopluk ; PITNER, Tomáš: Structured and interoperable logging for the cloud computing Era: The pitfalls and benefits. In: *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing* IEEE, 2013, S. 91–98
- [W3C06] W3C, World Wide Web C.: *The XMLHttpRequest Object*. <https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>, 2006. – [Online; abgerufen am 27.10.2020]
- [W3C20] W3C, World Wide Web C.: *About W3C Standards*. <https://www.w3.org/standards/about.html>, 2020. – [Online; abgerufen am 29.10.2020]
- [Wat18] WATERHOUSE, Peter: *Monitoring and Observability — What's the Difference and Why Does It Matter? – The New Stack*. <https://thenewstack.io/monitoring-and-observability-whats-the-difference-and-why-does-it-matter/>, 2018
- [WC14] WALKER, Jeffrey D. ; CHAPRA, Steven C.: A client-side web application for interactive environmental simulation modeling. In: *Environmental Modelling & Software* 55 (2014), S. 49–60
- [YM20] YOU, Evan ; MITWIRKENDE individuelle: *Vue.js*. <https://vuejs.org/>, 2020. – [Online; abgerufen am 12.10.2020]
- [ZHF<sup>+</sup>15] ZHU, Jieming ; HE, Pinjia ; FU, Qiang ; ZHANG, Hongyu ; LYU, Michael R. ; ZHANG, Dongmei: Learning to log: Helping developers make informed logging decisions. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* Bd. 1 IEEE, 2015, S. 415–425

- [Zip21] ZIPKIN AUTOREN: *Jaeger: open source, end-to-end distributed tracing*. <https://zipkin.io/>, 2021. – [Online; abgerufen am 11.02.2021]