

Bachelorthesis

Nachvollziehbarkeit von Nutzerinteraktionen und Anwendungsverhalten am Beispiel JavaScript-basierter Webanwendungen

An der Fachhochschule Dortmund
im Fachbereich Informatik
Studiengang Software- und Systemtechnik
Vertiefung Softwaretechnik
erstellte Bachelorthesis
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. März 2021

Kurzfassung

Im Projektalltag führen Probleme in JavaScript-basierten Webanwendungen zu teuren Ausfällen, die durch fehlende Informationen oftmals später behoben werden als vergleichbare Fehler in Backendsystemen. Aus diesem Grund wird in dieser Arbeit das Problem der Nachvollziehbarkeit bei Webanwendungen dieser Art untersucht.

Das Ziel der Arbeit ist es, herauszufinden, warum Entwickler und Betreiber von einer Webanwendung weniger relevante Informationen erhalten. Weiterhin ist zu ergründen, wie dieser Informationslücke entgegenwirkt werden kann. Wie sehen hierbei bewährte Ansätze aus und wie ist der Stand der Technik. Letztendlich ist ein Proof-of-Concept zu erstellen, anhand dessen die gefundenen Ansätze anzuwenden und zu veranschaulichen sind.

Um das Ziel der Arbeit zu erreichen, wurde zunächst die Ausgangssituation beschrieben, um festzustellen welche besonderen Eigenschaften diese Webanwendungen besitzen. Folgend wurde die Nachvollziehbarkeit als solche ergründet und welche Methoden existieren, um eine bessere Nachvollziehbarkeit zu erreichen. Darauf aufbauend beleuchtet die Arbeit zudem den Stand der Technik, auf Basis dessen einige Technologien kriteriengeleitet ausgewählt wurden, um diese beim Proof-of-Concept zu verwenden.

Der Proof-of-Concept wurde auf Basis einer Demoanwendung erstellt, welche eine mit Angular erstellte SPA ist. Diese SPA stellt einen Checkout-Wizard dar, welcher eingebaute Fehlerszenarien enthält. Für den Proof-of-Concept wurde diese SPA in verschiedenen Aspekten erweitert, um bspw. Logs, Metriken, Traces und Fehler zu protokollieren. Diese Daten wurden an Partnersysteme (Splunk, Jaeger, LogRocket) weitergeleitet und somit den Entwicklern sowie Betreibern aufbereitet zur Verfügung gestellt.

Letztendlich wurde die erstellte Lösung durch zuvor definierte Anforderungen und Fehlerszenarien überprüft, dabei wurde festgestellt, dass der geforderte Mehrwert erreicht werden konnte. Weiterhin konnte eine Übertragbarkeit auf andere ähnliche Softwareprojekte aufgezeigt werden, sodass die erstellte Lösung auch hier anwendbar ist.

Abstract

Failures in web applications that rely on JavaScript often result in costly outages. The time to fix can also be magnified compared to similar issues with backend applications, caused by missing crucial information needed to debug and fix the issue. Therefore, this thesis explores the problem of observability of web applications.

The goal of the work is therefore to bring forth an understanding, why relevant information is missing. Furthermore, it is to be explored, how this information can be obtained and if there are tried and tested methods to achieve this. Based on this gathered understanding, a proof of concept is to be developed, that uses the findings and illustrates their purpose.

To achieve the goal of the work, first and foremost the environment is investigated, e. g. which characteristics of web applications distinguish from backend applications. After that, the term “observability“ is defined and examined. The state of the art regarding “observability“ is studied, resulting in the knowledge of current methods and technologies. Based on these findings, some technologies are selected for use in the proof of concept.

The proof of concept is developed based on a demo application, that is in its core an Angular SPA. The SPA is a wizard for a webshop checkout, which also contains built-in issues. This demo application was then extended to yield more information, such as logs, metrics, traces and errors. These data was then forwarded to systems, that utilized them to enable developers and operators to “observe“ the SPA. These systems were Splunk, Jaeger and LogRocket.

Finally, the created solution was verified based on requirements and its ability to reveal crucial information about the built-in issues. It was found that the solution yielded the required information and also met the vast majority of previously defined requirements. Additionally, it was determined that the solution is applicable to similar software projects.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zielsetzung	2
1.2.1	Abgrenzung	2
1.3	Vorgehensweise	3
1.4	Open Knowledge GmbH	3
2	Ausgangssituation	4
2.1	Browserumgebung	4
2.1.1	Browserprodukte	4
2.2	JavaScript	5
2.2.1	Content-Security-Policy	6
2.2.2	Cross-Origin Resource Sharing (CORS)	6
2.3	Rich-Internet-Applications	7
2.3.1	Single-Page-Applications	8
2.4	Nachvollziehbarkeit/Observability	9
2.4.1	Nachvollziehbarkeit bei SPAs	9
2.4.1.1	Logdaten	9
2.4.1.2	Fernzugriff	10
3	Methoden und Praktiken	11
3.1	Methoden für eine bessere Nachvollziehbarkeit	11
3.1.1	Fehlerberichte	11
3.1.2	Die Grundpfeiler der Observability	12
3.1.2.1	Logging	12
3.1.2.2	Metriken	12
3.1.2.3	Tracing	13
3.1.3	OpenTelemetry	15
3.1.4	Application-Performance-Monitoring (APM)	17
3.1.4.1	Infrastructure-Monitoring (IM)	17
3.1.4.2	Application-and-Service-Monitoring (ASM)	18
3.1.4.3	Real-User-Monitoring (RUM)	18

3.1.4.4	Error-Monitoring	18
3.1.4.5	Distributed-Tracing	18
3.1.5	Log-Management	19
3.1.6	Session-Replay	19
3.2	Werkzeuge und Technologien	20
3.2.1	Recherche	20
3.2.2	Übersicht	21
3.2.3	Kategorisierung	22
3.2.4	Vorauswahl	25
3.2.5	Kriterien	26
3.2.6	Bewertung und Auswahl	27
3.2.7	Vorstellung der Technologien	30
3.2.7.1	Splunk	30
3.2.7.2	Jaeger	31
3.2.7.3	Sentry	32
3.2.7.4	LogRocket	34
4	Erstellung Proof-of-Concept	35
4.1	Vorstellung der Demoanwendung	35
4.1.1	Verhaltensdefinition	35
4.1.2	Backend	39
4.1.3	Frontend	40
4.1.3.1	Warenkorb	41
4.1.3.2	Rechnungsadresse	42
4.1.3.3	Lieferdaten	43
4.1.3.4	Zahlungsdaten	44
4.1.3.5	Bestellübersicht	45
4.1.3.6	Bestellbestätigung	46
4.1.4	Fehlerszenarien	46
4.1.4.1	„Keine Übersetzungen“	47
4.1.4.2	„Ungültige Adressen sind gültig“	47
4.1.4.3	„Lange Verarbeitung“	47
4.1.5	Repräsentation	47
4.2	Anforderungen	48
4.2.1	Definitionen	48
4.2.2	Anforderungsanalyse	49
4.2.3	Anforderungsliste	50
4.2.3.1	Funktionsumfang	50
4.2.3.2	Eigenschaften	52
4.2.3.3	Partnersysteme	53
4.3	Konzept	55
4.4	Implementierung	57

Inhaltsverzeichnis

4.4.1	Backend	57
4.4.2	Frontend	58
4.4.2.1	Datenweiterleitung im „Backend4Frontend“	58
4.4.2.2	Traces und Metriken	59
4.4.2.3	Logging	62
4.4.2.4	Fehler	63
4.4.2.5	Weiterleitung an Splunk	64
4.4.2.6	Session-Replay	67
4.4.3	Architektur	68
5	Ergebnis	70
5.1	Demonstration	70
5.1.1	Aufdecken des Szenarios „Keine Übersetzungen“	70
5.1.2	Aufdecken des Szenarios „Ungültige Adressen sind gültig“	72
5.1.3	Aufdecken des Szenarios „Lange Verarbeitung“	72
5.2	Bewertung der Anforderungen	73
5.3	Übertragbarkeit	75
5.4	Einfluss für den Nutzer	76
6	Abschluss	79
6.1	Zusammenfassung	79
6.2	Fazit	79
6.3	Ausblick	79
	Anhang	80
7	Anhang	80
7.1	Studien zur Browserkompatibilität	80
7.2	Weitere Demonstration	81
7.2.1	LogRocket	81
7.2.2	Splunk	81
7.2.3	Jaeger	83
	Eidesstattliche Erklärung	84
	Abkürzungsverzeichnis	85
	Abbildungsverzeichnis	86
	Tabellenverzeichnis	88
	Quellcodeverzeichnis	88

Literaturverzeichnis

90

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 seinen Kunden Versicherungsprodukte ausschließlich im Direktvertrieb, d. h. vor allem über das Internet und zusätzlich auch über Telefon, Fax oder Brief an. Im Unterschied zum klassischen Versicherer verfügen Direktversicherer 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 serverseitigem Rendering realisiert wurden, sind heutzutage JavaScript-basierte Webanwendungen mit clientseitigem Rendering die weit verbreitet [WC14]. Bei einer solchen Webanwendung befindet sich ein Großteil der Anwendungslogik im Browser des Nutzers, z. B. erfolgt bei einem Wizard erst bei Absenden eine Interaktion mit einem Server.

Im produktiven Einsatz kommt es auch bei gut getesteten Webanwendungen vor, dass bspw. unvorhergesehene Fehler in der Berechnung oder Verarbeitung auftreten. Liegt die Ursache für den Fehler im Browser, z. B. aufgrund einer ungültigen Wertkombination, steht der Betreiber vor einer Herausforderung. Bei Server-Anwendungen können Fehlermeldungen in Log-Dateien geprüft werden, ein Betreiber einer Webanwendung hat i. d. R. keine Möglichkeit die notwendigen Informationen über den Nutzer und seine Umgebung abzurufen. Hinzu kommt, dass der Betreiber gar nicht mitbekommt, wenn ein Nutzer auf 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 ist es, den Betreibern einer JavaScript-basierten Webanwendung die Möglichkeit zu geben, das Verhalten ihrer Anwendung und die Interaktionen von Nutzern nachzuvollziehen. Diese soll insbesondere bei Fehlerfällen gewährleistet sein, ist aber auch in anderen Situationen erstrebenswert, 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) ist hingegen nicht vorgesehen. Daraus ergibt sich die Forschungsfrage:

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

Zur Vorbereitung der Entwicklung des Proof-of-Concepts ist der Stand der Technik zu recherchieren. Die daraus resultierenden Erkenntnisse sind bei der Erstellung zu beachten und anzuwenden.

Die anzustrebende Lösung ist ein Proof-of-Concept, welcher eine, zu erstellende, Demoanwendung erweitert. Die Demoanwendung steht repräsentativ für eine JavaScript-basierte Webanwendung, bei der die zuvor benannten Hürden zur Nachvollziehbarkeit bestehen.

Weiterhin gilt es zu beleuchten, welche Auswirkungen der Einsatz von Methoden zur Nachvollziehbarkeit für Nutzer der Webanwendung hat. Wird bspw. die Leistung der Webanwendung beeinträchtigt (erhöhte Ladezeit, erhöhte Datenlast).

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

1.2.1 Abgrenzung

Die Demoanwendung wird als Single-Page-Application (SPA) [JSD15] realisiert. Bei der Datenerhebung und -verarbeitung werden datenschutzrechtliche Aspekte nicht näher zu betrachten. Darüber hinaus liegt bei der Recherche zum Stand der Technik eine allgemein bewertende Gegenüberstellung von Technologien nicht im Fokus der Arbeit.

1.3 Vorgehensweise

Für die Realisierung eines Proof-of-Concepts zur Erhöhung der Nachvollziehbarkeit einer bestehenden Anwendung, wird zunächst die theoretische Seite des Forschungsfeldes beleuchtet. Hierzu gehört eine nähere Betrachtung der Umgebung „Browser“, von Webanwendungen weiter gilt es den der „Nachvollziehbarkeit“ zu definieren und im Hinblick auf SPAs zu erläutern.

Darauf aufbauend werden aktuelle Ansätze zur verbesserten Nachvollziehbarkeit recherchiert und beschrieben. Speziell sollen hierbei übergreifende und allgemein angewandte Methoden ausgearbeitet und beschrieben werden. Weiterhin ist darauffolgend der Stand der Technik aus Wirtschaft und Literatur zu recherchieren und vorzustellen.

Bevor der Proof-of-Concept implementiert wird, soll ein Konzept erstellt werden, welches darlegt, wie der Proof-of-Concept eine verbesserte Nachvollziehbarkeit erreicht. Folgend ist auf Basis des Konzeptes die Demoanwendung zu erweitern, welches den Proof-of-Concept darstellt. 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 Leiter des Standortes Essen, Dipl.-Inf. Stephan Müller, übernimmt die Zweitbetreuung. Neben Stephan Müller ist Christian Wansart ein Stakeholder der hier zu erstellenden Lösung, er ist bei Open Knowledge angestellt.

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 gegründet und beschäftigt heute über 80 Mitarbeiter. Mitte 2017 wurde in Essen der zweite Standort eröffnet, an dem derzeit 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. Trotz seiner Größe hat Open Knowledge eine starke Außendarstellung in Fachpublikationen sowie auf Fachkonferenzen. So sind sowohl die Geschäftsführer als auch diverse Mitarbeiter von Open Knowledge bspw. als Redner auf der Javaland oder als Autoren wie dem Java Magazin vertreten.

2 Ausgangssituation

2.1 Browserumgebung

Web-Browser haben sich seit der Veröffentlichung von Mosaic, einem 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. Mittels JavaScript können Entwickler heutzutage komplexe browser-unabhängige Webanwendungen realisieren [JSD15]. Diese Entwicklung hat den Browser als populäre Laufzeitumgebung für Anwendungen über alle Betriebssysteme und Plattformen hinweg etabliert [TM11].

2.1.1 Browserprodukte

Die Vielfalt an Browsern bereitet Webentwicklern immer wieder Herausforderungen bei dem Ziel ein einwandfrei funktionierendes Produkt, unabhängig der Browserpräferenz des Nutzers, bereitzustellen. 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 [W3C21], 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 ist. Dies spricht dafür, dass das Problem von XBIs weniger präsent ist als zuvor. Somit wird die besondere Hürde, die XBIs darstellen, in dieser Arbeit als nicht relevant angesehen.

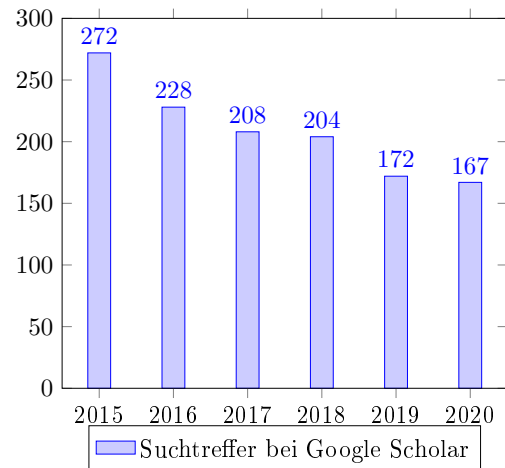


Abb. 2.1: Studien zur Browserkompatibilität, eigene Darstellung (vgl. Abschnitt 7.1)

Im Jahr 2020 gab es eine wesentliche Entwicklung, die die Kompatibilität zwischen Browsern erhöhen und Unterschiede zukünftig spürbar reduzieren wird. Microsoft ist beim Folgeprodukt zum Internet Explorer, dem Microsoft Edge, von einer proprietären Browser-Engine zu Chromium gewechselt [Mic20b] und verwendet somit denselben Kern wie die Browser Chrome und Opera. Zudem stellte Microsoft zum 30.11.2020 den Support für den Internet Explorer 11 ein [Mic20a].

Eine Übersicht der aktuellen Browser mit Marktanteil und den verwendeten Engines für Layout und JavaScript, kann in Tabelle 2.1 betrachtet werden. Hierbei wird deutlich, dass der überwiegende Anteil (>95%) der Desktop-Browser auf den Engines Blink, WebKit und Gecko bzw. für JavaScript auf V8, JavaScriptCore und SpiderMonkey basieren.

Browser	Anteil	Layout-Engine	JavaScript-Engine
Chrome	66,47%	Blink	V8
Safari	10,27%	WebKit	JavaScriptCore
Firefox	8,17%	Gecko	SpiderMonkey
Edge	8,01%	Blink	V8
Opera	2,68%	Blink	V8
Internet Explorer	1,89%	Trident	Chakra

Tab. 2.1: Übersicht aktueller Browser [Sta21f] [DD20]

2.2 JavaScript

Als JavaScript 1997 veröffentlicht und in den NetScape Navigator integriert wurde, gab es 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 des Browsers schränkt den Zugriff auf das Dateisystem ein. Darüber hinaus sind auch keine Zugriffe auf native Bibliotheken oder die Ausführung von nativem Code möglich [OKSK15]. Um z. B. Daten beim Client zu speichern oder auch Videos abzuspielen bieten Browser eigene Schnittstellen hierfür an.

1999 nahm Microsoft im Internet Explorer 5.0 eine neue Funktion in ihre JavaScript-Umgebung auf: Asynchronous JavaScript and XML (Ajax) [MM21a]. 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 von Ajax durch das W3C [W3C06] das Konzept Anklang bei Entwicklern [Dog20] [IF14] und ist seitdem der Grundstein für unser dynamisches und interaktives Web [DCZ11].

Durch die Unterstützung von Ajax wurden Webanwendungen immer beliebter. Entwickler beklagten jedoch vermehrt, dass Browser die Abfragen von JavaScript nur auf dem bereitstellenden Webserver, also „same-origin“, erlauben [Ran09]. Um eine größere Flexibilität zu ermöglichen, wurde im selben Jahr der Standardisierung von Ajax ein erster Entwurf zur Absicherung beim 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 im Browser auszuführen.

2.2.1 Content-Security-Policy

Im Browser haben Entwickler die Möglichkeit zu bestimmen, welche Funktionalitäten einer Webanwendung zur Verfügung stehen sollen und wie diese vom Browser einzuschränken sind. Diese Funktion heißt Content-Security-Policy (CSP) und dient unter anderem dem Schutz vor Cross-Site-Scripting. Mit der CSP kann eine Webanwendung beschränken, welche Funktionalitäten in JavaScript verfügbar sind und von wo aus Skripte und Daten geladen werden dürfen [MM21d]. Weiterhin kann eingerichtet werden, dass bei einem Versuch diese CSP-Regeln zu umgehen, darüber Bericht erstattet wird.

2.2.2 Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) [OPBW06] entwickelte sich aus dem Wunsch von Entwicklern, nicht auf einen einzelnen Webserver beschränkt zu sein. Eine Einschränkung, die eingeführt wurde, um Nutzer vor Missbrauch zu schützen [MHDJ09]. CORS lockert diese Einschränkung unter Berücksichtigung sicherheitskritischer Aspekte. Das Konzept von CORS stellt sicher, dass aus JavaScript heraus keine Daten von Webservern angefragt werden können, welche nicht explizit der Anfrage zustimmen [MM21e].

Wie eine „cross-origin“ Ajax-Anfrage nach dem Konzept von CORS gehandhabt wird, ist in Abbildung 2.2 dargestellt. Wenn eine HTTP-Anfrage nicht „simple“¹ 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 erlaubt ist, wird die eigentliche Ajax-Anfrage ausgeführt. Ansonsten schlägt die Anfrage fehl und wird in JavaScript aus Sicherheitsgründen ohne detaillierte Beschreibung gemeldet [MM21e].

¹„simple“ ist sie, wenn 1. die Methode GET, HEAD oder POST entspricht; 2. keine benutzerdefinierten Header enthalten sind; und 3. bei POST-Anfragen der „Content-Type“ einem dieser Werte entspricht: „application/x-www-form-urlencoded“, „multipart/form-data“ oder „text/plain“ [MM21e].

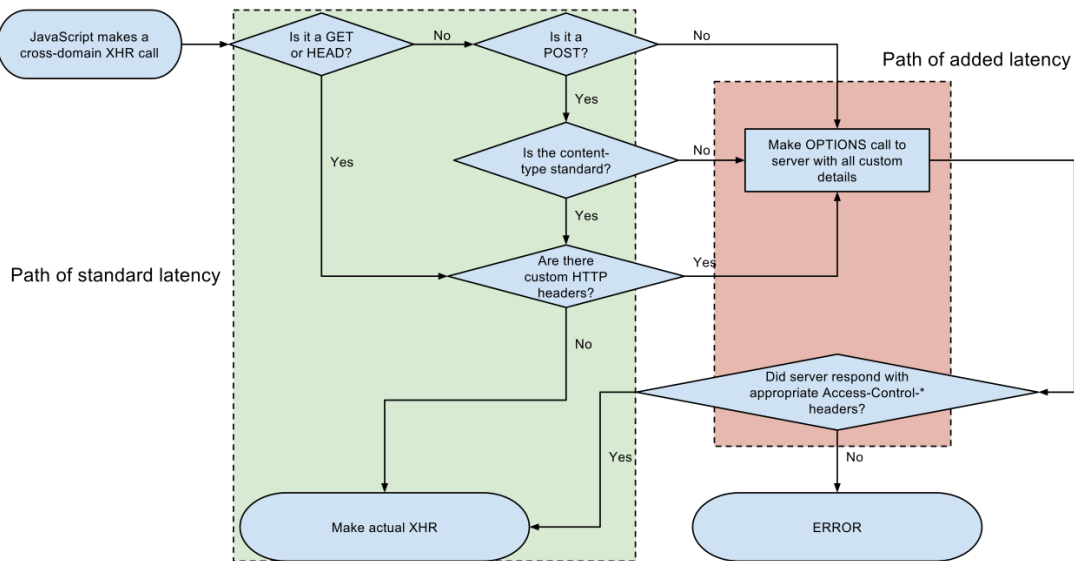


Abb. 2.2: Flowchart über den Ablauf von Ajax-Anfragen mit CORS [Blu15]

2.3 Rich-Internet-Applications

Rich-Internet-Applications (RIA, oder auch Rich-Web-Application) werden oftmals damit assoziiert, dass sie Webanwendungen darstellen, welche Merkmale und Funktionalitäten einer Desktopanwendung besitzen [CGM14] [PLSC05]. Sie besitzen bspw. erweiterte Interaktionsmöglichkeiten (wie Drag-And-Drop), eine detail- und funktionsreiche Benutzeroberfläche mit Fokus auf Benutzbarkeit, zudem eine erhöhte Responsiveness im Vergleich zu klassischen Webanwendungen [CGM14].

Erste Ansätze von RIAs gab es bereits ohne eine ausreichende Unterstützung von JavaScript in Browsern. 2002 wurde das Produkt Macromedia Flash MX veröffentlicht, eine an Macromedia Flash (später Adobe Flash) angelehnte Laufzeitumgebungen, die speziell für die Erstellung von RIAs entwickelt wurde [All02], welche zudem den Begriff der Rich-Internet-Application prägte [CGM14]. Weiterhin wurden RIAs mit einer Vielzahl von Technologien umgesetzt, wie z. B. mit Macromedia Flex (später Adobe Flex nun Apache Flex) oder Java Applets [PLSC05] [FN07] [FRSF10] [IF14]. Diese RIAs benötigen extra Umgebungen, die meist via Plugins in den Browser eingebunden werden, ein Beispiel hierfür ist Adobe Flash. Adobe stellte jedoch 2020 den Support für Flash ein [Men19]. Heutzutage sind aber JavaScript-basierte RIAs [FN07], auch Rich-Web-Based-Applications genannt [DD18] [Men19], die Norm.

Neben den Vorteilen einer RIA, besitzen diese jedoch auch einige Nachteile. RIAs in jedweder Form stellen eine Herausforderung für Webcrawler dar und erschweren bzw. verhindern die Indexierung der Seite durch Suchmaschinen [CDM⁺12]. RIAs leiden zudem oft darunter, dass sie die Funktionen zur Barrierefreiheit in Browsern nicht nutzen. Um dem entgegenzuwirken veröffentlichte das W3C 2011 eine Empfehlung [W3C20], um die Barrierefreiheit auch bei RIAs zu gewährleisten.

2.3.1 Single-Page-Applications

Single-Page-Applications (SPA) stellen eine spezielle Art von Rich-Internet-Applications dar. Sie gehen bei der dynamischen Inhaltsdarstellung einen Schritt weiter [JSD15]: Die gesamte Anwendung wird über ein einziges HTML-Dokument und die darin referenzierten Inhalte erzeugt. Eine Charakteristik sind nicht nur erweiterte Interaktionen, sondern auch komplexe lokale Zustände, die im Client gepflegt werden. Wird beispielsweise eine neue Seite aufgerufen, wird statt einer Dokumentenabfrage via HTTP ein interner Zustand geändert, welcher dann DOM-Manipulationen auslöst, die die Darstellung der Seite ändern.

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 [Goo21b], React [Fac21] oder Vue.js [YM21].

SPAs bieten zudem durch ihre stark clientbasierte Herangehensweise 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].

Dieser grundsätzlich andere Ansatz bringt mit sich auch negative Eigenschaften. Unter anderem werden native Browserfunktionen umgangen, wie die automatisch befüllte Browserhistorie, da keine HTML-Dokumente angefragt werden. 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 Angular, React und Vue.js teils spezielle Implementierungen oder es muss bspw. die jeweilige Router-Bibliothek herangezogen werden.

Nichtsdestotrotz ist ein jahrelanger Trend des Erfolges von Single-Page-Applications zu erkennen [JSD15]. Zudem stehen Entwicklern heutzutage eine große Auswahl an erprobten Technologien in diesem Gebiet zur Verfügung, um umfangreiche und nahtlos funktionierende SPAs zu erstellen [GB21].

2.4 Nachvollziehbarkeit/Observability

Neben der Umgebung Browser und SPAs als solche, stellt die Nachvollziehbarkeit einen wichtigen Bestandteil dieser Arbeit dar. Nachvollziehbarkeit bedeutet allgemein, dass über ein resultierendes Verhalten eines Systems auch interne Zustände nachvollzogen werden können. Dabei handelt es sich nicht um eine neue Idee, sondern sie fand bereits 1960 im Gebiet der Kontrolltheorie starke Bedeutung [Ká60]. Nach Freedman [Fre91] und Scrocca *et al.* [STM⁺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 Monitorings 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.

2.4.1 Nachvollziehbarkeit bei SPAs

In dieser Arbeit wird die Nachvollziehbarkeit bei Webanwendungen näher betrachtet. Wie zuvor in Unterabschnitt 2.3.1 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.

2.4.1.1 Logdaten

Ähnlich wie bei anderen Umgebungen gibt es eine standardisierte Log- bzw. Konsolenausgabe für die JavaScript-Umgebung in Browsern [MM21b]. Diese Ausgabe ist dem Standard-Benutzer i. d. R. unbekannt, daher kann nicht erwartet werden, dass Nutzer im Fehlerfall ein Log bereitstellen können. Durch die zuvor beschriebenen Härtungsmaßnahmen von Browsern ist es hinzukommend nicht möglich, das Log direkt in eine Datei zu schreiben.

Um die Logdaten im Browser erheben zu können, gilt es entweder ein spezielles Log-Framework in der Webanwendung zu verwenden bzw. die bestehende Schnittstelle zu überschreiben oder zu wrappen und die Logdaten an ein Partnersystem weiterzuleiten, welches die Sammlung, Aggregation und Auswertung ermöglicht.

2.4.1.2 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, wie z. B. das Remote-Application-Debugging [Ora21] von Java, die für diesen Zweck eingesetzt werden kann. Weiterhin sind bei einer Webanwendung, die für den offenen Markt geschaffen wurde, die Nutzer i. d. R. zahlreich und unbekannt, sodass sich eine solche Funktionalität nicht realistisch umsetzen lässt.

3 Methoden und Praktiken

3.1 Methoden für eine bessere Nachvollziehbarkeit

Nachvollziehbarkeit bietet einen wichtigen Mehrwert für Entwickler und Betreiber von Webanwendungen. Wie aber kann eine verbesserte Nachvollziehbarkeit bei einer Webanwendung erreicht werden? In diesem Abschnitt werden einige Methoden vorgestellt mit denen dieses Ziel erreicht werden kann.

3.1.1 Fehlerberichte

Fehlerberichte sind ein klassisches Mittel, um den Nutzer aufzufordern selber Bericht über ein Problem zu erstatten (vgl. Abbildung 3.1). Hiermit können Fehler, aber auch unverständliche Workflows, aufgedeckt werden. Weiterhin können Informationen des Nutzers ermittelt werden, wie es hierzu gekommen ist und warum es ein Problem darstellt, vorausgesetzt er gibt dies an. Fehlerberichte sind ein einfach einzusetzendes Mittel, welches keine oder keine starke Integration in sonstige Teile der Anwendung benötigt.

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

Abb. 3.1: Fehlerbericht in der Instagram App [Fac20]

3.1.2 Die Grundpfeiler der Observability

Da Fehlerberichte nicht ausreichend sind, um den Entwicklern eine angemessene Nachvollziehbarkeit zu gewährleisten, sind zusätzliche Konzepte notwendig, um dies zu erreichen. Nach Sridharan *et al.* [Sri18] sowie [SM21] [LVG⁺18] [PCQ⁺18] existieren drei Grundpfeiler der Observability, die in ihrer Funktion einzigartig sind und sich gegenseitig ergänzen: Logging, Metriken und Tracing.

3.1.2.1 Logging

Logging bezeichnet die systematische Protokollierung von Softwareprozessen und ihren internen Zuständen [ZHF⁺15]. Diese Protokolle, auch Logs genannt, 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 ausreichend detaillierte Logmeldungen in die Anwendung integriert sind und sie verstanden werden können [ZHF⁺15].

Logs stellen die primäre oder in manchen Fällen die einzige Methode dar, wie Betreiber und Entwickler das Verhalten einer Anwendung in einer Produktivumgebung nachvollziehen können [CCP12] [ZHF⁺15]. Gerade in Fehlersituationen können Logs kritische Informationen bereitstellen. Wie in Unterunterabschnitt 2.4.1.1 beschrieben ist die Erhebung von Logmeldungen in einer JavaScript-basierten Webanwendung nicht trivial und findet daher kaum Anklang.

Logmeldungen erfolgen meist textbasiert und in einem menschenlesbaren Format. Wenn nun ein Aggregator 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 definierten Format erzeugt, i. d. R. in Form von Key-Value Paaren. Durch die feste Definition des Formates, z. B. als JSON-Objekt, wird bei der Loganalyse ermöglicht, effizient die notwendigen Daten zu extrahieren.

Structured-Logging ermöglicht, dass auf Basis der Protokolle komplexe Datenanalysen durchgeführt werden können [TVNP13]. Mit diesen lassen sich auch bei großen Datenmengen situationsrelevante Informationen entlocken [LGB19]. So lassen sich auch aus Logmeldungen gesonderte Informationen wie Metriken extrahieren.

3.1.2.2 Metriken

Metriken sind numerische Repräsentationen von Daten, die in einer Zeitspanne aufgenommen wurden. Mithilfe von Metriken können mathematische Konzepte verwendet werden, um Verständnis zu gewinnen und Vorhersagen zu treffen [Sri18]. Metriken sind

zudem optimal für effiziente Datenbankabfragen sowie eine Langzeitspeicherung geeignet. Dies ist durch die einheitliche Struktur und die hauptsächlich numerischen Werte bedingt.

Beispielsweise identifiziert Prometheus [Pro21] Metriken über einen eindeutigen Namen und Schlüsselwertpaare (vgl. Abbildung 3.2) und speichert die assoziierten Daten mit einem Zeitstempel und einem Fließkommawert. Zudem können Technologien rund um die Metrikerhebung einfacher mit großen Datenmengen umgehen, da die Metriken aggregierbar sind [Sri18].

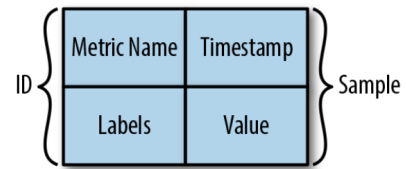


Abb. 3.2: Struktur eines Prometheus-Metriken-Datensatzes [Sri18]

Durch die Kompatibilität, mathematische Konzepte auf Metriken anwenden zu können, eignen sie sich zudem dafür, die Verfügbarkeit von Komponenten überprüfbar zu gestalten und erlauben so eine Übersicht der „Gesundheit“ eines Gesamtsystems zu veranschaulichen [PCQ⁺18] [Sri18]. Metriken besitzen jedoch auch Grenzen in dem was sie aufdecken können, nämlich lässt sich nur mithilfe von Metriken nicht das Verhalten eines Systems zu einem exakten Zeitpunkt nachvollziehen. Hierbei helfen wiederum Logs und bieten dabei detaillierte Einsicht in einzelnen Situationen. Damit die Kommunikation zwischen Komponenten oder Systemen nachvollziehbar wird, besonders aus Sicht eines einzelnen ursprünglichen Aufrufs, sind weder Logs noch Metriken ausreichend zielführend - aus diesem Grund entwickelte sich das Tracing [PCQ⁺18] [Sri18].

3.1.2.3 Tracing

Tracing beschäftigt sich mit dem Aufzeichnen von Kommunikationsflüssen in Software-systemen [OHH⁺16]. Hierbei werden die Kommunikationsflüsse innerhalb einer Anwendung bzw. innerhalb eines Systems erfasst. Tracing zeichnet aber auch die Kommunikationsflüsse bei verteilten Systemen auf, um diese, meist komplexen Zusammenhänge, zu veranschaulichen. Das Tracing von verteilten Systemen wird als „Distributed-Tracing“ bezeichnet. Ein herstellerunabhängiger Standard, der sich aus diesem Gebiet entwickelt hat, ist OpenTracing [Ope20b].

OpenTracing bildet diese Kommunikationsflüsse über zwei grundlegende Objekte ab: Traces und Spans. Ein Span besitzt einen Anfangs- und einen Endzeitpunkt und *umspannt* meist eine Methodenaufruf. Bei einer Webanwendung kann dies eine JavaScript-Funktion oder ein durch den Nutzer hervorgerufener Eventfluss sein. Werden innerhalb eines Spans weitere Spans erstellt, z. B. durch einen Methodenaufruf, dann werden diese als Kindspans des ursprünglichen Spans assoziiert. 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 3.3) oder auch über die zeitliche Reihenfolge der einzelnen Spans (vgl. Abbildung 3.4).

Ein verteilter Trace, oftmals „Distributed-Trace“ genannt, ist ein Trace, der sich aus den Spans verschiedener Systeme zusammensetzt, die miteinander kommunizieren. Hierbei werden die Traceinformationen über zusätzliche Felder bei existierenden Aufrufen propagiert, wie z. B. dem Einfügen eines Trace-Headers bei HTTP-Anfragen. Die dann an ein Tracesystem gemeldeten Spans gehen somit über die Grenzen von Anwendungen, Prozessen und Netzwerken hinaus und bilden einen Distributed-Trace [Ope21f]. Auf Basis von realen Aufrufen können die tatsächlichen Zusammenhänge der einzelnen Systeme miteinander nachempfunden werden.

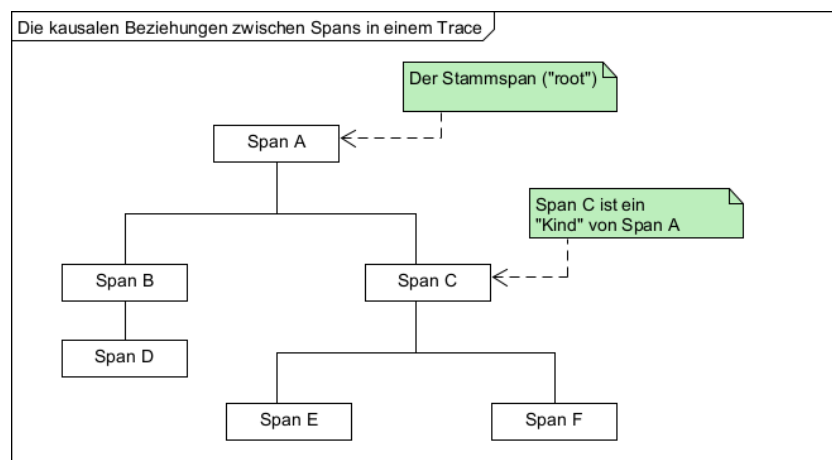


Abb. 3.3: Kausale Beziehung zwischen Spans. Eigene Darstellung.

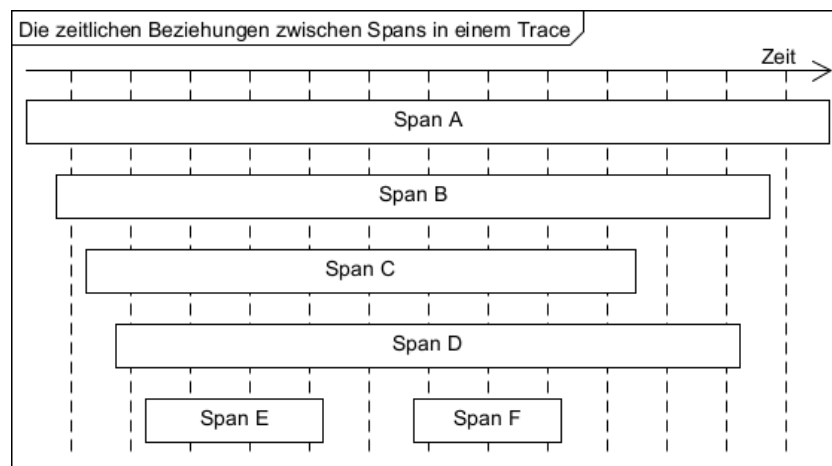


Abb. 3.4: Zeitliche Beziehung zwischen Spans. Eigene Darstellung.

3.1.3 OpenTelemetry

Auf Basis dieser der drei Grundpfeiler Logging, Metriken und Tracing haben sich einige Technologien entwickelt. Viele dieser Ansätze sind proprietär und nicht miteinander kompatibel, weswegen das Bedürfnis einer Standardisierung entstand. Um die hier angesiedelten Technologien zu vereinheitlichen entstanden u. A. OpenTracing, OpenCensus [Ope21a] sowie OpenTelemetry, die jeweils darauf abzielen herstellerunabhängige Observability-Konzepte zu definieren.

OpenTelemetry (OTel) ist ein sich derzeit¹ entwickelnder Standard, welcher als Ziel hat, das Erfassen, Weiterleiten und Verarbeiten von Tracing-, Metrik- und Logdaten² herstellerunabhängig zu ermöglichen. OTel entwickelte sich aus dem Zusammenschluss der Teams hinter den beiden Standards OpenTracing und OpenCensus [Jos19]. Microsoft, Google, führende Unternehmen und Entwickler von Observability-Technologien sowie die Cloud-Native-Computing-Foundation (CNCF) arbeiten an der Entwicklung des OTel-Standards [PSM⁺20] [Ope20a]. OTel versucht nicht nur die bisherige Landschaft zu vereinigen, sondern definiert eine zukunftsorientierte Architektur, die aus unterschiedlichen Komponenten besteht, und definiert wie diese miteinander kommunizieren [PSM⁺20].

Der OpenTelemetry-Standard definiert einige Komponenten, die spezielle Aufgabengebiete abdecken und standardisiert mit anderen Komponenten kommunizieren. Diese Komponenten werden nachfolgend anhand des Beispiels eines Tracing-Spans sowie der Abbildung 3.5 erläutert.

- **API:** Die API stellt die öffentlich sichtbare Schnittstelle der „low-level“ OTel-Verarbeitung (des SDKs) dar, die Zielgruppe umfasst Entwickler sowie instrumentierende Bibliotheken. Mit der API kann ein Entwickler einen Trace initialisieren, darauf aufbauend einen Span erzeugen und diesen Span starten.
- **Instrumentation-Library:** Bei dieser Bibliothek handelt es sich um eine spezifische Anbindung der OTel-API bspw. an ein Framework (wie JAX-RS oder Angular). Teilweise erlauben solche Bibliotheken auch eine automatische Erfassung der Daten, sodass bei relevanten Methoden (wie Schnittstellauufrufen) ein Span erzeugt wird.
- **SDK:** Das SDK stellt das Herzstück der Logik bei OTel dar und beinhaltet die interne Verarbeitung der Daten. Ein erzeugter Span wird hier mit seinen Kontextinformationen zwischengespeichert und bei Beendigung des Spans wird dieser an angebundene Exporter übergeben.

¹Ein erster (General-Availability-)Release der Spezifikation ist für die erste Hälfte 2021 geplant [Ope21d] (Stand 01.03.2021)

²Die Entwicklung einer Logging-Spezifikation ist im Gange [Ope21e].

- **Exporter:** Exporter sind spezifische Anbindungen, welche Daten im OTEL-Format annehmen und für Gegenstellen aufbereiten und an diese transportieren. Die Gegenstelle kann entweder eine Datensenke darstellen oder auch eine weiterverarbeitende Komponente sein. Im Beispiel (vgl. Abbildung 3.5) wird der Span nicht in ein anderes Format überführt, da er an einen OTEL-Collector gesendet wird.
- **Collector:** Kollektoren sind eine OTEL-Schnittstelle, um unterschiedliche OTEL-Daten anzunehmen und diese an weitere Systeme mithilfe von Exportern zu übergeben. Bei dem Exporter in diesem Beispiel werden die Daten in das passende Format des Telemetry-Backends überführt.
- **Telemetry-Backend:** Ein Telemetry-Backend stellt die Datensenke der OTEL-Daten dar und bietet den Entwicklern und Betreibern eine Visualisierung der gesammelten Daten. Beispiele hierfür sind z. B. Jaeger [Jae21b] oder Prometheus [Pro21].

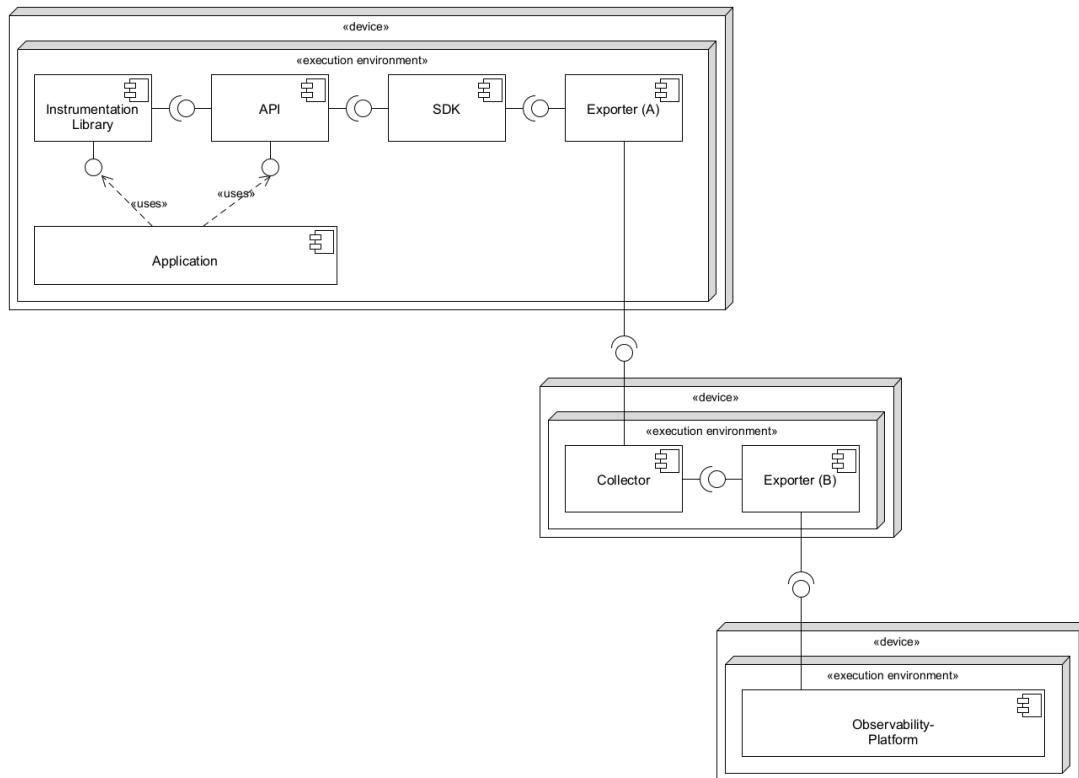


Abb. 3.5: Komponenten von OpenTelemetry, eigene Darstellung auf Basis von [Ope21c]

3.1.4 Application-Performance-Monitoring (APM)

In der Praxis haben sich einige Technologien entwickelt und etabliert, welche die Nachvollziehbarkeit von Anwendungsverhalten und Nutzerinteraktion ermöglichen oder verbessern. Auf Basis der zuvor vorgestellten Methoden sowie neuer Ansätze haben sich in der Wirtschaft eine Reihe von Praktiken entwickelt. Einer dieser Ansätze ist das Application-Performance-Monitoring (APM, teils auch -Management).

Eine exakte Definition von APM lässt sich nicht nennen, denn es existiert kein Konsens, welche Eigenschaften und Funktionen ein APM umfasst. Ahmed *et al.* [ABC⁺16], Heger *et al.* [HHMO17], Rabl *et al.* [RSJ⁺12] sowie Dynatrace [Dyn20] definieren, dass ein APM eine Menge an Methoden, Techniken und Werkzeugen umfasst, die das System konstant überwachen und Aufschluss über den Zustand geben, sodass die Verfügbarkeit des Systems sichergestellt werden kann. Eine andere Definition vertreten Santos Filipe [Fil20], New Relic [New20] und Gartner [Gar20], welche APM weniger allgemeingültig, sondern über explizite Teilaspekte definieren, die es erfüllen muss, um als konformes APM bezeichnet zu werden. Aber auch innerhalb dieser Definition findet sich kein Konsens bzgl. der zu erfüllenden Aspekte, damit ein Monitoring-System zu einem APM wird.

In dieser Arbeit wird auf Basis der ersten und eher allgemeingültigen Definition ein APM definiert: Ein APM befasst sich mit dem Beobachten eines Softwaresystems und der Gewinnung von relevanten Daten aus diesem System zur näheren Analyse, um Rückschlüsse auf die Gesundheit des Systems zu ermöglichen und so die Verfügbarkeit sicherzustellen. Um dies zu erreichen, lassen sich 5 Fachgebiete einteilen, die unterschiedliche Aspekte eines Softwaresystems aufdecken [FEH⁺14] [Gar20] [YZS16]:

1. Infrastruktur-Monitoring (IM)
2. Application-and-Service-Monitoring (ASM)
3. Real-User-Monitoring (RUM)
4. Error-Monitoring
5. Distributed-Tracing

3.1.4.1 Infrastructure-Monitoring (IM)

Infrastructure-Monitoring beschäftigt sich hauptsächlich mit der Überwachung der Infrastruktur. Hierbei wird die Verfügbarkeit von Netzwerkressourcen sowie die Auslastung von Hard- und Softwareressourcen überwacht. Dieses Monitoring kann ohne Anpassungen der Software erfolgen und stellt somit ein Beispiel für Black-Box-Monitoring dar [Fil20]. Zum Beispiel ist die Überwachung von CPU- und Speicherausnutzung eines Systems ein Teil des Infrastructure-Monitoring.

3.1.4.2 Application-and-Service-Monitoring (ASM)

Bei Application-and-Service-Monitoring (ASM) handelt es sich um White-Box-Monitoring. Das bedeutet, dass die Softwarekomponenten angepasst werden müssen, sodass innerhalb der Laufzeitumgebungen Daten gesammelt werden können. Beispielsweise werden die Antwortzeit von Schnittstellauufrufen protokolliert und systematisch überwacht. Auf Basis der Daten lassen sich Abweichungen, von einzelnen Systemen oder vom aktuellen Gesamtsystem zu einem vorherigen Zeitpunkt feststellen.

3.1.4.3 Real-User-Monitoring (RUM)

Real-User-Monitoring beschäftigt sich mit dem Mitschneiden von Nutzerinteraktionen und Umgebungseigenschaften einer Benutzeroberfläche [CGL⁺15]. Um diese Daten zu ermitteln ist eine Änderung der Software für die Benutzeroberfläche notwendig, welches RUM zu einem White-Box-Monitoring macht. RUM wird jedoch nicht dazu verwendet, die Interaktionen eines einzelnen Nutzers aufzudecken, sondern Aufschluss über die gesamte Nutzerschaft der Anwendung zu erhalten. Die Daten werden oftmals nach den Interaktionen oder auch nach Umgebungseigenschaften, wie dem Browser der Nutzer gruppiert. Dadurch lassen sich Probleme bei der User-Experience [OTMC11], aber auch Performanceprobleme der Anwendung feststellen und sowie deren Ursache (z. B. die Umgebung des Nutzers) [CGL⁺15].

3.1.4.4 Error-Monitoring

Error-Monitoring konzentriert sich auf das Erfassen und Melden von Fehlern [BT19]. Es lässt sich sowohl als White-Box- sowie als Black-Box-Monitoring umsetzen, da über existierende Protokollierung bereits Fehler festgestellt werden können. Hierzu kann es sinnvoll sein eine Software anzupassen, also White-Box-Monitoring einzusetzen, um mehr Kontextinformationen zu den Fehlern zu erfassen. Error-Monitoring wird oftmals eng mit Issue-Management verbunden, um aufgetretene Fehler und deren Behebung nachzuhalten [BT19].

3.1.4.5 Distributed-Tracing

Beim Distributed-Tracing handelt es sich um die fortgeschrittene Art des Tracings, welches systemübergreifend den Ablauf von Abfragen protokolliert (vgl. Unterunterabschnitt 3.1.2.3). Diese Art von Monitoring gibt, anders als die zuvor beschriebenen Arten, keine Einsicht in einzelne Komponenten, sondern veranschaulicht die resultierenden Interaktionen einer Abfrage.

3.1.5 Log-Management

Neben dem APM gibt es zudem weitere Funktionalitäten, wie z. B. das Log-Management, die Technologien in der Praxis vorweisen. Log-Management umfasst die Erfassung, Speicherung, Verarbeitung und Analyse von Logdaten von Anwendungen. Neben diesen Funktionen bieten solche Werkzeuge oftmals fundierte Suchfunktionen und Visualisierungsmöglichkeiten [RAF19]. Um die Daten aus einer Anwendung heraus zu exportieren, gibt es meist eine Vielzahl an Integrationen für Frameworks und Logbibliotheken.

Einer der wichtigsten Aspekte des Log-Managements, ist die Fähigkeit mit großen Datenmengen umzugehen und dabei den Nutzern zu ermöglichen mit diesen Daten zu arbeiten, Analysen durchzuführen und auch alte Datensätze abrufen zu können [CSP12]. Damit die teils enormen Datenmengen den Entwicklern und Betreibern zur Verfügung stehen, aber gleichzeitig nicht das System beeinträchtigen, sind spezielle Architekturkonzepte erforderlich. Beispielsweise werden selten abgerufene oder alte Daten in einen Langzeitspeicher überführt, welcher für die Speicherung optimiert ist, aber im Gegenzug keine zeitlich effizienten Ergebnisse liefern kann [CSP12].

3.1.6 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. Um eine realitätsnahe und entsprechend hilfreiche Nachstellung zu erstellen, sind viele Aspekte aufzuzeichnen. Ein realitätsnahes Session-Replay zeichnet eine enorme Datenmenge für jede Nutzersitzung auf und benötigt besonders beim Einsatz in Browsern eine effiziente Kommunikation, um die User-Experience (UX) nicht negativ zu beeinflussen [ČF17] [Log21b].

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 von und zu der Eventloop mitgeschnitten (vgl. Abbildung 3.6).

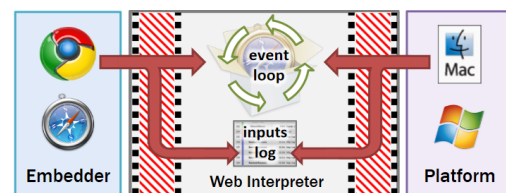


Abb. 3.6: Mitschneiden von DOM-Events, Abb. aus [BBKE13]

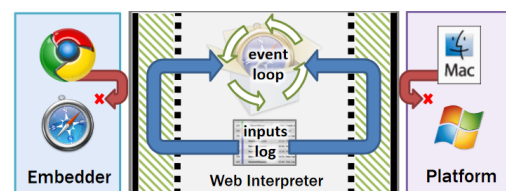


Abb. 3.7: Abspielen von DOM-Events, Abb. aus [BBKE13]

Beim Abspielen werden die Aufrufe dann in derselben Reihenfolge an die Eventloop übergeben (vgl. Abbildung 3.7). Somit erfolgt eine exakte Wiederholung einer Sitzung, welches eine detaillierte Nachvollziehbarkeit ermöglicht. Leider benötigt dieser Ansatz eine gepatchte Version von WebKit [App21b], weswegen auch Zugriff auf das System des Endnutzers vorausgesetzt wird. Zur Veröffentlichung des Berichtes bedeutete dies, dass die Browser Safari, Chrome und Opera unterstützt wurden - jedoch benutzt heute nur noch der Safari Webkit. Aufgrund der Notwendigkeit den Browser der Nutzer zu modifizieren und der Tatsache, dass es seit mehr als 5 Jahren³ nicht mehr gepflegt wird, scheidet Timelapse für die hier angestrebte Lösung aus. Die vorgestellten Konzepte stellen jedoch nützliche Kernprinzipien für das Session Replay im Allgemeinen dar.

3.2 Werkzeuge und Technologien

Um die gewünschte Lösung, den Proof-of-Concept, zu erstellen, muss zuvor der Stand der Technik erörtert werden. In diesem Abschnitt wird ein Überblick über aktuelle Technologien gegeben. Die vorgestellten Technologien werden kategorisiert und nach zuvor definierten Kriterien bewertet. Daraus ergibt sich eine Auswahl, der für den Proof-of-Concept in Frage kommenden Technologien.

3.2.1 Recherche

Um relevante und aktuelle Technologien zu ermitteln, wurde neben verfügbarer Literatur auch auf etablierte Plattformen bei der Gegenüberstellung von Technologien gesetzt. Dabei wurden Gartner⁴ und StackShare⁵ eingesetzt. Die identifizierten Technologien werden im nachfolgenden Abschnitt vorgestellt.

Mithilfe von Gartners „Magic Quadrant for APM“ [Gar20] konnte festgestellt werden, dass folgende Werkzeuge zu den führenden Technologien in der Kategorie angehören: *AppDynamics* [App21a], *Dynatrace* (ehemals ruxit) [Dyn21], *New Relic* [New21], *Broadcom DX APM* [Bro21], *Splunk APM* [Spl21b] sowie *Datadog* [Dat21]. Bestätigt werden einige dieser Technologien in der Bewertung bei StackShare [Sta21d]. Hier sind insbesondere New Relic und Datadog zu erwähnen oft sowie die Application Insights [Mic21b] des *Azure Monitors* von Microsoft.

In der Literatur finden sich auch Vergleiche und Empfehlungen, so fanden z. B. Martínez *et al.* [HMLJ21] in ihrer Evaluierung von Werkzeugen bei der Unterstützung von E2E-Tests, dass die beiden OpenSource-Technologien *Jaeger* [Jae21b] und *Zipkin* [Zip21] aktiv

³Timelapse GitHub Repo <https://github.com/burg/replay-staging/>

⁴Gartner ist ein global agierendes Forschungs- und Beratungsunternehmen im Bereich der IT [Ivy13]

⁵StackShare (<https://stackshare.io>) ist eine Vergleichsseite für Entwicklerwerkzeuge und Technologien, die auf Basis von Nutzereingaben Vergleiche erzeugt [KKV19]

dabei helfen können Fehlerszenarien in Microservice-Architekturen besser nachzuvollziehen. Weiterhin beschrieben Li *et al.* [LLG⁺19], wie mit *Prometheus* [Pro21], Jaeger, Zipkin und *Fluentd* [Flu21a] eine Datenanalyse von Microservices ermöglicht werden kann. Hinzukommend beschreiben Picoreti *et al.* [PCQ⁺18] eine Observability-Architektur, die auf *Fluentd*, *Prometheus* und *Zipkin* basiert.

Bei StackShares Gegenüberstellung von Error-Monitoring-Produkten [Sta21b] stechen drei Technologien heraus: *Sentry* [Fun21a], *TrackJS* [Tra21] sowie *Rollbar* [Rol21]. Die zwei erst genannten waren zudem auch bei der Gegenüberstellung der Monitoring-Lösungen [Sta21c] gelistet.

StackShare bezeichnet Session-Replay als „User-Feedback-as-a-Service“. Hierbei [Sta21e] lassen sich ebenfalls drei etablierte Produkte identifizieren: *Inspectlet* [Ins21], *FullStory* [Ful21] und *LogRocket* [Log21a]. Während *Inspectlet* und *FullStory* hauptsächlich auf die Nachvollziehbarkeit von User-Experience abzielen, konzentriert sich *LogRocket* auf technische Informationen, die für Entwickler von Bedeutung sind [FČ18]. Gartner bietet zudem eine Übersicht [Gar21] über Produkte im „Web and Mobile App Analytics Market“ an, in der sich *Google Analytics* [Goo21a], *Adobe Analytics* [Ado21] sowie *LogRocket* auf den obersten Positionen befinden.

3.2.2 Übersicht

In der Tabelle 3.1 werden nachfolgend die gefundenen Technologien näher veranschaulicht. Auf Basis der Produktbeschreibungen der Hersteller wird untersucht, welche Funktionalität die jew. Technologie vorweisen kann. Genauer werden folgende, zuvor identifizierte, Funktionalitäten unterschieden und den zugeordnet: IM, ASM, RUM, Error-Monitoring, Log-Management, (Distributed-)Tracing sowie Session-Replay. Um den Funktionsumfang zur jew. Funktionalität anzugeben, werden folgende 4 Schlüssel verwendet:

1. **ja**: Die Funktionalität ist vorhanden und der Funktionsumfang entspricht der Definition.
2. **ja(*)**: Die Funktionalität ist vorhanden, aber sie ist nicht so umfangreich wie bei anderen Technologien.
3. **eingeschränkt**: Die Funktionalität ist nur unter bestimmten Voraussetzungen vorhanden oder ist nur teilweise implementiert.
4. **keine Angabe**: Die Funktionalität ist nicht vorhanden.

Technologie	IM	ASM	RUM	Error-Monitoring	Log-Mgmt.	Tracing	Session-Replay
Adobe Analytics	eing.	eing.	ja(*)	eingeschr.			
Airbrake	ja	ja		ja			
AppDynamics	ja	ja	ja(*)	ja	eingeschr.	ja	
Azure Monitor	ja	ja		ja	ja	ja	
Broadcom DX APM	ja	ja		teils	ja	ja	
Bugsnag				ja			
DataDog	ja	ja	ja(*)	ja	ja	ja	
Dynatrace	ja	ja	ja(*)	ja	ja	ja	
Elastic Stack	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Fluentd					eingeschr.		
FullStory			ja	teils			ja
Google Analytics	eing.	eing.	ja(*)	eingeschr.			
Graylog	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Inspectlet			ja	teils			ja
Jaeger						ja	
LogRocket			ja	ja			ja
New Relic	ja	ja	ja(*)	ja	ja	ja	
Papertrail	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Prometheus	ja	ja					
Raygun	ja	ja	ja(*)	ja			
Rollbar			ja(*)	ja			
Sentry			ja(*)	ja			
Splunk APM (Signal-FX)	ja	ja		ja		ja	
Splunk Enterprise	ja(*)	ja(*)	ja(*)	ja(*)	ja		
TrackJS			ja(*)	ja			
Zipkin						ja	

Tab. 3.1: Übersicht der untersuchten Technologien

3.2.3 Kategorisierung

Um die Veranschaulichung übersichtlicher zu gestalten, werden die Technologien auf Basis gemeinsamer Funktionalitäten kategorisiert. Auf Basis der evaluierten Funktionalitäten ergaben sich 6 Kategorien, in die die Technologien eingeordnet werden können. Diese Kategorien werden folgend erläutert:

1. APM-Plattformen

Zu APM-Plattformen gehören allen voran Technologien, bei denen das Application- and-Service-Monitoring sowie das Infrastructure-Monitoring Kernfunktionalitäten darstellen. Bis auf ein Werkzeug begrenzen sich keine APM-Plattform nur auf diese beiden Aspekte, sondern können meist mehrere andere Funktionalitäten vorweisen. Am häufigsten sind Aspekte des Error-Monitorings, des Log-Managements sowie eines Distributed-Tracings vorzufinden. Neben technischen Aspekten bilden viele dieser Tools mithilfe von ASM und RUM auch Einsichten in die geschäftliche Leistung der Anwendung. Auf Basis von RUM wird teils Nutzerverhalten gruppiert visualisiert, um die Nutzerschaft besser verstehen zu können - eine Ansicht einer einzelnen Nutzersitzung, wie beim Session-Replay, ist jedoch nicht Teil dessen.

2. Log-Plattformen

Als Log-Plattformen werden alle Technologien bezeichnet, die eine Verarbeitung von Logdaten als ihre Kernfunktionalität verstehen. Nahezu alle hier angesiedelten Werkzeuge sind in der Lage Entwicklern und Betreibern eine detaillierte Analyse der Logdaten zu ermöglichen. Weiterhin steht oftmals die Funktionalität zur Verfügung diese Daten auch visuell Darstellungen zu können. Auf Basis der analytischen Funktionalitäten können zudem Aspekte von IM, ASM, RUM oder Error-Monitoring nachgestellt werden. Neben diesen Funktionalitäten steht aber auch ein effizientes Persistenzkonzept im Vordergrund, damit mit den enormen Datenmengen aus unterschiedlichen Systemen umgegangen werden kann [HZH⁺17].

3. Distributed-Tracing-Systeme

Hiermit werden jene Technologien beschrieben, die ein Distributed-Tracing ermöglichen. Effiziente Architektur stehen oftmals im Vordergrund, welche explizit auf die enormen Datenmengen angepasst ist, die beim Distributed-Tracing anfallen können [SBB⁺10].

4. Error-Tracking

Die Kategorie Error-Tracking zeichnet sich dadurch aus, dass Technologien die Erhebung und Visualisierung von Fehlerdaten als ihre Kernfunktionalität verstehen. Weiterhin besitzen viele dieser Werkzeuge ein detailliertes Issue-Management, mit dem sich Teams organisieren können, um Fehler zu beheben und Arbeiten nachzuhalten.

5. Session-Replay-Dienste

Die Technologien der Kategorie Session-Replay zeichnen Nutzersitzungen auf und stellen diese Betreibern und Entwicklern in nachgestellter Videoform bereit. Hierbei lässt sich eine geschäftliche und eine technische Repräsentation unterscheiden. Bei ersterem werden Nutzersitzungen teils gruppiert und als Heatmaps dargestellt,

bei letzterem werden detaillierte technische Informationen mitgeschnitten und dargestellt [FČ18].

6. Web-Analytics

Die letzte Kategorie beschäftigt sich mit Web-Analytics-Technologien. Diese befassen sich mit der Evaluierung der Performance einer Webanwendung, sei es im geschäftlichen oder auch im technischen Sinne [PSF04] [Kau07]. Allgemeiner lässt sich anhand der Charakteristika sagen, dass Web-Analytics eine sehr spezifische Untermenge von APM-Plattformen darstellt.

In der Tabelle 3.2 werden die Technologien gruppiert nach ihrer Kategorie dargestellt. Da nicht alle Kategorien gleich hilfreich für die hier angestrebte Lösung sind, findet im Unterabschnitt 3.2.4 eine Vorauswahl statt, welche Funktionskategorien näher betrachtet werden sollen.

Technologie	IM	ASM	RUM	Error-Monitoring	Log-Mgmt.	Tracing	Session-Replay
APM-Plattformen							
AppDynamics	ja	ja	ja(*)	ja	eingeschr.	ja	
Dynatrace	ja	ja	ja(*)	ja	ja	ja	
New Relic	ja	ja	ja(*)	ja	ja	ja	
Broadcom DX APM	ja	ja		teils	ja	ja	
Splunk APM (Signal-FX)	ja	ja		ja		ja	
DataDog	ja	ja	ja(*)	ja	ja	ja	
Azure Monitor	ja	ja		ja	ja	ja	
Prometheus	ja	ja					
Log-Plattformen							
Papertrail	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Elastic Stack	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Fluentd					eingeschr.		
Splunk Enterprise	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Graylog	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Distributed-Tracing-Systeme							
Jaeger						ja	
Zipkin						ja	
Error-Tracking							
Sentry			ja(*)	ja			
TrackJS			ja(*)	ja			
Rollbar			ja(*)	ja			
Airbrake	ja	ja		ja			

Technologie	IM	ASM	RUM	Error-Monitoring	Log-Mgmt.	Tracing	Session-Replay
Bugsnag				ja			
Raygun	ja	ja	ja(*)	ja			
Session-Replay-Dienste							
Inspectlet			ja	teils			ja
FullStory			ja	teils			ja
LogRocket			ja	ja			ja
Web-Analytics							
Google Analytics	eing.	eing.	ja(*)	eingeschr.			
Adobe Analytics	eing.	eing.	ja(*)	eingeschr.			

Tab. 3.2: Kategorisierung der untersuchten Technologien

3.2.4 Vorauswahl

Wie zuvor beschrieben eignen sich nicht alle Funktionskategorien teils mehr teils weniger für die in dieser Arbeit angestrebte Lösung: Ein Proof-of-Concept, welches die Nachvollziehbarkeit einer Webanwendung von Anwendungsverhalten und Nutzerinteraktionen für **Betreiber und Entwickler** verbessert.

APM-Plattformen bieten durch ihr IM und ASM aufschlussreiche Einsichten in die Leistung und Verfügbarkeit einer Anwendung, helfen aber nicht oder kaum bei der Aufdeckung von einzelnen Problemen. Sie bieten hilfreiche Informationen aus wirtschaftlicher und operativer Sicht, bieten aber technische Informationen. Ein weiterer Grund gegen die Nutzung von Werkzeugen des Application-Performance-Monitorings ist die hier existierende Marktmacht von proprietären Lösungen. Die proprietären Ansätze zeichnen sich durch vorgefertigte Lösungen aus, die aber unflexibel in den Anpassungsmöglichkeiten sind. Um diese Diskrepanz näher zu untersuchen wurden New Relic und Dynatrace evaluiert. Hierbei konnte festgestellt werden, dass die bereitgestellten Informationen, aus Sicht eines Entwicklers, nicht ausreichend Aufschluss boten. So fehlte eine detaillierte Einsicht in einzelne Fehlerszenarien oder auch Nutzersitzungen. Stattdessen wurden gruppierte Informationen bereitgestellt, die die grobe „Gesundheit“ des Systems widerspiegeln. Aus diesen Gründen, gerade aufgrund der divergierenden Zielgruppen, werden APM-Plattformen nicht näher betrachtet.

Aus einem ähnlichen Grund, wird die Kategorie Web-Analytics ebenso nicht näher behandelt. Werkzeuge im Web-Analytics-Bereich legen den Fokus sehr stark auf Überprüfung von wirtschaftlichen und operativen Eigenschaften, nicht jedoch den in dieser Arbeit erwünschten Zielen. Die übrig gebliebenen Kategorien werden im nächsten Unterabschnitt näher betrachtet und kriteriengeleitet bewertet.

3.2.5 Kriterien

Um für den Proof-of-Concept passende Technologien zu identifizieren, werden diese auf Basis verschiedener Kriterien bewertet. Diese Bewertung stützt sich auf öffentlich verfügbare Informationen, die die Hersteller der jeweiligen Technologie selber veröffentlicht haben. Die Kriterien werden folgend näher beschrieben.

1. Kostenfrei

Mit dem Kriterium „Kostenfrei“ soll bewertet werden, ob eine kostenfreie Variante dieser Technologie existiert. Existiert eine kostenfreie Variante, wird **ja** angegeben und ansonsten **nein**. Existiert jedoch eine kostenfreie Version, die entweder von den Funktionalitäten oder der zeitlichen Nutzung beschränkt ist, wird diese mit **f. beschränkt** bzw. **z. beschränkt** angegeben.

2. Support für Webanwendungen

Bewertet, ob eine Unterstützung für das Senden von Daten von Webanwendungen existiert. Dies kann in der Form einer Anbindung (bsw. ein Agent⁶) oder einer Schnittstelle sein. Ist eine Schnittstelle vorhanden, die aber nicht direkt aus einem Browserkontext ansprechbar ist, so ist diese Technologie mit **möglich** zu bewerten. Ist jedoch keine Schnittstelle vorhanden, die das Senden von eigenen Daten ermöglicht, so ist die Technologie mit **nein** zu bewerten.

3. OnPremise und SaaS

Mit diesen zwei Kriterien soll bewertet werden, wie diese Technologie eingesetzt werden kann. Ist sie in einer eigenen Infrastruktur aufsetzbar, also „On-Premise“, oder existiert die Technologie als buchbarer Dienst z. B. in der Cloud, also als Software-as-a-Service (SaaS).

4. Standardisierung

Setzt die jeweilige Technologie auf etablierte Standards, wie OpenTracing bei Distributed-Tracing? Falls nicht, sind einzelne Komponenten (z. B. zur Instrumentalisierung) quelloffen oder öffentlich spezifiziert, sodass diese ausgetauscht oder angepasst werden können.

5. Multifunktional

Mit dem Kriterium „Multifunktional“ wird bewertet, ob eine Technologie neben der Kernfunktionalität auch weitere Funktionalitäten aufweist. Eine Technologie ist multifunktional, wenn sie min. zwei nicht nah verwandte Funktionalitäten nahezu vollständig besitzt (**ja** oder **ja(*)**). Nah verwandt sind hierbei IM und ASM.

⁶Ein Agent ist eine Bibliothek, die die jeweiligen Daten (wie Klicks, Ladezeiten für Ressourcen und DOM-Events, usw.), eigenständig sammelt und an ein Partnersystem überträgt [RSJ⁺12]

6. Zielgruppe

Es ist einzuordnen, für welche Zielgruppen die Technologien einen Mehrwert bietet. Es werden folgende Zielgruppen differenziert: **Projektmanager**, **Fachabteilung** und **Entwickler**.

Auf Basis dieser Kriterien werden nachfolgend die Technologien bewertet. Anschließend erfolgt auf dessen Basis für jede Kategorie eine Auswahl.

3.2.6 Bewertung und Auswahl

In der Kategorie „Log-Plattformen“ findet sich auf Basis der Bewertung wenig Varianz zwischen den Technologien (vgl. Tabelle 3.3), lediglich Fluentd sticht hervor. Dies ist dadurch erklärbar, dass Fluentd kein vollständiges Log-Management darstellt, sondern es sich um einen Logaggregator handelt [Flu21b]. Somit ist Fluentd nur als verwandte Technologie anzusehen und fällt somit als Präferenz aus. Der Elastic-Stack eignet sich durch die hohe Flexibilität und der Komponente Logstash auch dazu, Log-Management mit ihr zu betreiben [Vet20] [RAF19]. Papertrail, Splunk sowie Graylog lassen sich als klassische Log-Management-Werkzeuge verstehen, da sie speziell auf diese Funktionskategorie angepasst sind. Graylog sowie der Elastic-Stack sind quelloffen, aber auch als SaaS verfügbar. Bei einem gewünschten OnPremise Einsatz kann lediglich nur Papertrail nicht eingesetzt werden, denn dies wird nicht unterstützt. Letztendlich lässt sich sagen, dass keine dieser vier Technologien ausschließende Eigenschaften besitzt, sondern allesamt für die in dieser Arbeit angestrebte Lösung geeignet sind.

Letztendlich wurde sich für **Splunk** entschieden, u. A. weil es beim Kunden bereits im Einsatz ist. Wie aber zuvor erwähnt, eignen sich die anderen Technologien ähnlich gut und eine erneute Auswahl mit anderen situationsbedingten Kriterien könnte variieren.

Technologie	Kostenfrei	Support f. Webanw.	On Premise	SaaS	Standard.	Multif.	Zielgruppe
Papertrail	f. begrenzt	eingeschr.	nein	ja	nein	ja	Fachabteilung, Entwickler
Elastic Stack	ja	eingeschr.	ja	ja	eingeschr.	ja	Fachabteilung, Entwickler
Fluentd	ja	eingeschr.	ja	ja	eingeschr.	nein	Entwickler
Splunk Enterprise	f. begrenzt	eingeschr.	ja	ja	nein	ja	Fachabteilung, Entwickler
Graylog	ja	eingeschr.	ja	ja	eingeschr.	ja	Entwickler

Tab. 3.3: Bewertung der Technologien der Kategorie „Log-Plattformen“

Im Gebiet der Distributed-Tracing-Systeme gibt es auch nur wenige oberflächliche Unterschiede (vgl. Tabelle 3.4). Sowohl Jaeger als auch Zipkin sind quelloffen und werden weitflächig eingesetzt [Hög20]. Jaeger scheint für neue Projekte attraktiver zu sein und findet dort mehr Einsatz, wie in StackShares Gegenüberstellung zu sehen ist [Sta21a]. Teilweise ist dies erklärbar durch die Ergebnisse, die Martínez *et al.* [HMQLJ21] herausfanden: Jaeger zeigt mehr hilfreiche Informationen an und kann diese schneller bereitstellen als Zipkin. Zudem entwickelt Jaeger aktiv eine Unterstützung des neuen OpenTelemetry-Standards [Jae21c], bei Zipkin findet sich keine vergleichbare Entwicklung. Aus diesen Gründen ist **Jaeger** hierbei das Werkzeug der Wahl.

Technologie	Kostenfrei	Support f. Webanw.	On Premise	SaaS	Standard.	Multif.	Zielgruppe
Jaeger	ja	eingeschr.	ja	nein	ja	nein	Entwickler
Zipkin	ja	eingeschr.	ja	nein	eingeschr.	nein	Entwickler

Tab. 3.4: Bewertung der Technologien der Kategorie „Distributed-Tracing-Systeme“

Die Auswahl in der Kategorie „Error-Tracking“ ist etwas diverser (vgl. Tabelle 3.5), denn hier weisen manche Technologien Funktionalitäten auf, die sonst in dem Gebiet fremd sind. Beispielweise bieten Airbrake und Raygun, neben einem Error-Monitoring, zudem Aspekte eines APM, sodass die Anwendung/das System auch im Normalbetrieb überprüft werden kann. Diese APM-Funktionalitäten sind jedoch nicht so ausgereift, wie bei spezialisierten APM-Lösungen. Airbrake und Raygun sind lediglich als SaaS-Produkte, während Sentry, Rollbar und Bugsnag auch als OnPremise-Lösung verfügbar sind. Sentry ist zudem quelloffen auf GitHub [Git21] veröffentlicht⁷ und entwickelt dort mit der Community an dem Produkt weiter. Weiterhin ist Sentry das einzige identifizierte Werkzeug, welches eine nicht zeitlich begrenzte Version der SaaS-Lösung zur Verfügung stellt. Eine aussagekräftige Entscheidung kann jedoch nicht getroffen werden, da alle Werkzeuge des Error-Monitoring die gewünschten Anforderungen ausreichend erfüllen. Dennoch wird sich an dieser Stelle für **Sentry** entschieden, auf Basis der zuvor nahe gelegten Gründe.

⁷Sentry GitHub Repo: <https://github.com/getsentry/sentry>

Technologie	Kostenfrei	Support f. Webanw.	On Premise	SaaS	Standard.	Multif.	Zielgruppe
Sentry	f. begrenzt	ja	ja	ja	eingeschr.	nein	Fachabteilung, Entwickler
TrackJS	z. und f. begrenzt	ja	nein	ja	nein	nein	Fachabteilung, Entwickler
Rollbar	z. und f. begrenzt	ja	ja	ja	nein	nein	Fachabteilung, Entwickler
Airbrake	z. und f. begrenzt	ja	nein	ja	nein	ja	Fachabteilung, Entwickler
Bugsnag	z. und f. begrenzt	ja	ja	ja	eingeschr.	nein	Fachabteilung, Entwickler
Raygun	z. und f. begrenzt	ja	nein	ja	nein	ja	Fachabteilung, Entwickler

Tab. 3.5: Bewertung der Technologien der Kategorie „Error-Tracking“

In der Beschreibung zur Kategorie „Session-Replay“ wurde erwähnt, dass einige dieser Werkzeuge eine eher geschäfts- und andere eher eine entwicklerorientierte Session-Replay-Funktionalität vorweisen (vgl. Tabelle 3.6). Genauer ist FullStory fast ausschließlich für das Nachvollziehen von User-Experience konzipiert, während LogRocket sehr detaillierte und sehr technische Informationen liefert [FČ18]. Inspectlet lässt sich als Mischung dieser beiden Sichten verstehen, bietet aber z. B. nicht alle Informationen an, die LogRocket darstellt [FČ18]. Da die hier angestrebte Lösung auf Betreiber und insbesondere Entwickler abzielt, fiel die Wahl auf **LogRocket**.

Technologie	Kostenfrei	Support f. Webanw.	On Premise	SaaS	Standard.	Multif.	Zielgruppe
Inspectlet	f. begrenzt	ja	nein	ja	nein	ja	Projektmanager, Fachabteilung, Entwickler
FullStory	f. begrenzt	ja	nein	ja	nein	ja	Projektmanager, Fachabteilung, Entwickler
LogRocket	f. begrenzt	ja	ja	ja	nein	ja	Fachabteilung, Entwickler

Tab. 3.6: Bewertung der Technologien der Kategorie „Session-Replay-Dienste“

3.2.7.2 Jaeger

Jaeger wurde 2017 als ein OpenSource-Projekt der CNCF gestartet [Jae21b]. Es ist ein System für verteiltes Tracing und bietet Funktionalitäten zur Datensammlung, –verarbeitung, und –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 befindet sich derzeit in der Entwicklung [Jae21c]. Darüber hinaus kann Jaeger genutzt werden, um Metriken nach Prometheus [Pro21] 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 3.10 zu sehen ist. Hierbei werden sowohl hierarchische 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, wie bspw. Logmeldungen oder Kontextinformationen, anzeigen.

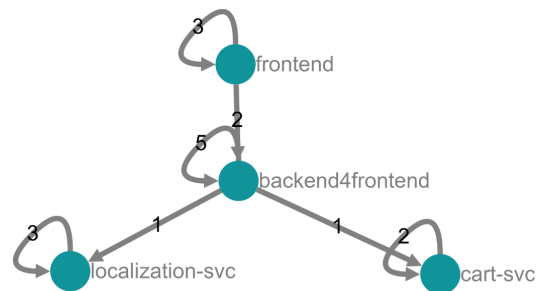


Abb. 3.9: Dienst-Abhängigkeits-Graph.
Quelle: Eigene Darstellung

Anhand der Traces generiert Jaeger zudem automatisch eine Netzwerk-Topology, anhand der die Beziehungen zwischen Diensten nachvollzogen werden können (vgl. Abbildung 3.9).

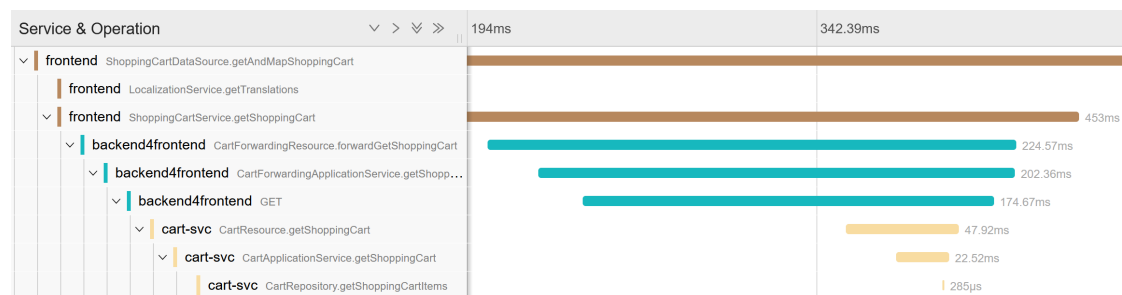


Abb. 3.10: Trace-Detailansicht. Eigener Screenshot aus Jaeger

3.2.7.3 Sentry

Sentry [Fun21a] 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, stellt Sentry auch eine unbegrenzt kostenlos nutzbare Version bereit, welche im Rahmen dieser Arbeit evaluiert wurde. Der Quellcode für das Backend von Sentry ist, wie zuvor beschrieben, quelloffen verfügbar. Darüber hinaus wird zudem eine OnPremise-Lösung angeboten, die auf Docker basiert [Fun21c]. Um von Webanwendungen Fehler zu erfassen und an Sentry zu melden, bietet Sentry bei NPM [npm21] quelloffene Pakete an [Fun21b]. Dabei werden u. A. Anbindungen für folgende Technologien bzw. Frameworks bereitgestellt: JavaScript, Angular, React und Vue.js.

Wird ein Fehler gemeldet, erstellt Sentry hierzu ein „Issue“, also einen Problembericht. In einem Problembericht finden sich detaillierte Informationen zum Fehler, wie den Stacktrace, den Zeitstempel, die Nutzerumgebung (Browser, Version, etc.) sowie ein Ausschnitt der zuletzt aufgetretenen Logmeldungen in der Browserkonsole (vgl. Abbildung 3.11). Zudem schneidet Sentry jegliche Nutzerinteraktionen mit und stellt diese in dem Problembericht dar (vgl. Abbildung 3.12). Treten Fehler gleichen Ursprungs auf, fasst Sentry diese im selben Problembericht zusammen. Weiterhin kann jeder einzeln aufgetretener Fehler näher betrachtet werden.

Die angebotenen Fehlerinformationen von Sentry sind zahlreich und helfen beim Nachvollziehen besser als Logs und Traces allein. Eine ganzheitliche Nachvollziehbarkeit kann Sentry jedoch nicht anbieten, da die Informationen nur im Fehlerfall erhoben werden. Somit reicht Sentry nicht allein aus, um die in dieser Arbeit erwünschte Nachvollziehbarkeit zu erreichen.

3 Methoden und Praktiken

The screenshot shows a Sentry issue page for a **TypeError** on `onCreateSubmit(main)`. The error message is "can't access property 'doSomething', var is undefined". The issue is assigned to "EST-22" and has 3 events and 1 user. The user is "marvin.kienitz@example.com" (ID: 4711). The event is "ad9257f45d21445aa27c94c677a7085b" from Feb 21, 2021, 12:29:32 PM UTC, with a JSON file (40.5 KB). The event details show a message: "There were 4 errors encountered while processing this event". The user's environment is Firefox 86.0 on Windows 10.

TypeError onCreateSubmit(main)

can't access property "doSomething", var is undefined

ISSUE # ...EST-22 EVENTS 3 USERS 1 ASSIGNEE

Resolve Ignore Share Open in Discover

Details Activity 0 User Feedback 0 Attachments Tags Events Merged Issues Similar Issues

Event [ad9257f45d21445aa27c94c677a7085b](#)
Feb 21, 2021 12:29:32 PM UTC | [JSON \(40.5 KB\)](#)

There were 4 errors encountered while processing this event [Show](#)

marvin.kienitz@example.com
ID: 4711

Firefox
Version: 86.0

Windows
Version: 10

Abb. 3.11: Kerninformation eines Issues. Eigener Screenshot aus Sentry

The screenshot shows the "BREADCRUMBS" section in Sentry. It contains a table with 5 columns: TYPE, CATEGORY, DESCRIPTION, LEVEL, and TIME. The table shows a sequence of user interactions: a click on a text input, an input change, another click on a submit button, and finally an exception (TypeError) that occurred at 12:28:42.

BREADCRUMBS

Filter By Search breadcrumbs...

TYPE	CATEGORY	DESCRIPTION	LEVEL	TIME
ui.click	ui.click	input#lastNameInput.form-control.ng-dirty.ng-valid.ng-touched[type="text"]	info	12:28:40
ui.input	ui.input	input#lastNameInput.form-control.ng-dirty.ng-valid.ng-touched[type="text"]	info	12:28:41
ui.click	ui.click	form.ng-dirty.ng-touched.ng-valid > button.btn.btn-primary[type="submit"]	info	12:28:42
exception	exception	TypeError: can't access property "doSomething", var is undefined	error	12:28:42

Abb. 3.12: Verlauf der Userinteraktionen. Eigener Screenshot aus Sentry

3.2.7.4 LogRocket

LogRocket [Log21a] ist ein SaaS-Produkt des gleichnamigen Unternehmens und konzentriert sich auf detailliertes Session-Replay von JavaScript-basierten Clientanwendungen, um Probleme zu identifizieren, nachvollziehen und lösen zu können. Neben des SaaS-Produktes bietet LogRocket für Unternehmenskunden auch eine OnPremise-Lösung an. Anders als vergleichbare Session-Replay-Technologien nicht das Marketingteam o. Ä. [FČ18] die primäre Zielgruppe, sondern Entwickler.

LogRocket bietet eine kostenlose Testversion des SaaS-Produktes an, welche für die Evaluierung verwendet wurde. Zur Datenerhebung wird das Paket `logrocket` bei NPM [npm21] angeboten, 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 3.13).

Neben dem JavaScript-SDK bietet LogRocket quelloffene Plugins für folgende Bibliotheken: Redux, React, MobX, Vuex, ngrx, React Native. Zusätzlich bietet LogRocket auch eine Integration für andere Tools, wie z. B. Sentry. Bei der Integration für Sentry wird bei einem gemeldeten Fehler in Sentry direkt auf das „Video“ in LogRocket verlinkt.

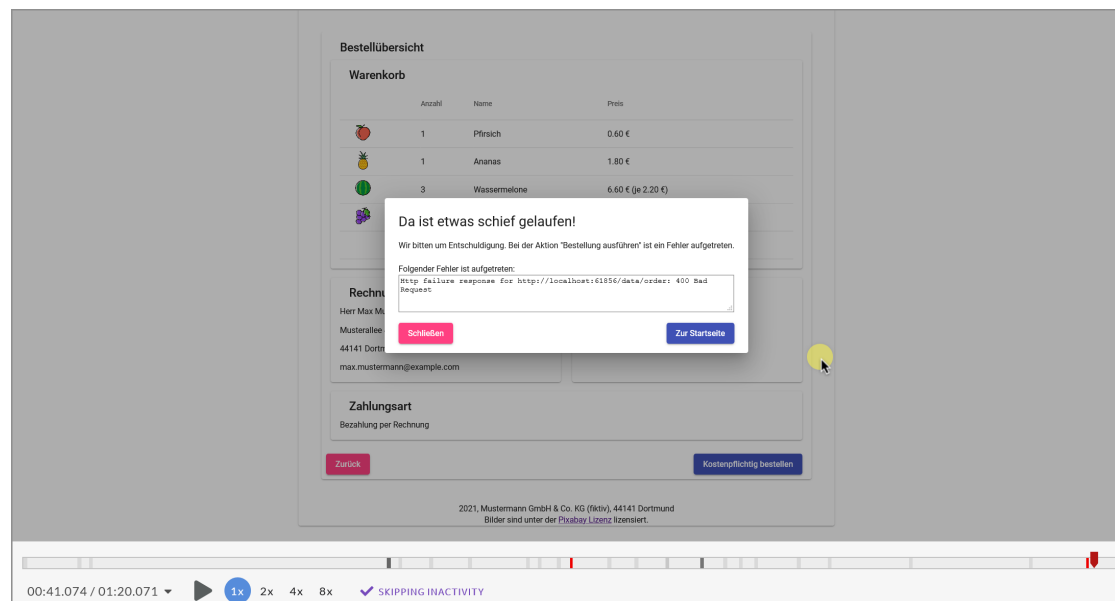


Abb. 3.13: Ausschnitt eines Session-Replays. Eigener Screenshot aus LogRocket.

4 Erstellung Proof-of-Concept

4.1 Vorstellung der Demoanwendung

Vor dem zu erstellenden Konzept wird zunächst eine Demoanwendung erstellt, an der das Konzept anzuwenden ist. Dieser Abschnitt beschäftigt sich mit der Vorstellung der Demoanwendung, wie diese aufgebaut ist und was für eine Art von Webanwendung sie repräsentiert.

Die Motivation nennt ein konkretes Problem eines Kunden der Open Knowledge. Um eine moderne Webanwendung [DD18] darzustellen, wird die Demoanwendung in Grundzügen den Aufbau der Webanwendung des Direktversicherers nachahmen. Die Webanwendung realisiert einen Wizard [RIS18], also eine 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.

Die Demoanwendung soll eine Bestellfunktionalität eines Obst-Webshops darstellen. Der Warenkorb hierfür wird anfangs dynamisch generiert und simuliert so, dass dieser durch einen vorgelagerten Prozess erstellt wurde. 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.

4.1.1 Verhaltensdefinition

Der gewünschte Funktionsumfang der Demoanwendung wurde mittels einer Verhaltensdefinition festgehalten. 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.

Die BDD-Definition wurde in der gängigen Gherkin-Syntax [Sma21] geschrieben. Die Syntax ist natürlich zu lesen. Nachfolgend 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
4       sollen die Artikel samt Name, Anzahl sowie Preis angezeigt werden.
5       Der Warenkorb stellt den Einstieg der Software dar.
6
7     Scenario: Kundin öffnet den Warenkorb
8       When die Kundin den Warenkorb öffnet
9       Then soll sie die ausgewählten Artikel mit Bild, Artikelnamen,
10        Anzahl und dem Gesamtpreis des Artikels sehen
11       And sie soll den Gesamtpreis für alle Artikel sehen
12
13    Scenario: Kundin soll zur nächsten Seite wechseln können
14      Given die Kundin hat die gewählten Produkte geprüft
15      When sie auf den "Bestellvorgang starten"-Button klickt
16      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
4       Rechnungsadresse eingeben können, welche die Pflichtfelder Anrede
5       , Vornamen, Nachnamen, Straße, Hausnummer, Postleitzahl sowie die
6       E-Mail-Adresse umfassen.
7
8     Scenario: Kundin kommt auf die Rechnungsadresse-Seite vom Warenkorb
9       aus
10      When die Rechnungsadresse-Seite zum ersten Mal aufgerufen wird
11      Then sollen die Eingabefelder leer sein
12
13    Scenario: Kundin kommt auf die Rechnungsadresse-Seite von der
14      Lieferadresse-Seite aus
15      Given die Kundin hatte bereits zuvor die Rechnungsadresse ausgefü
16      llt
17      Then sollen die zuvor eingegebenen Adressdaten weiterhin vorhanden
18      sein
19
20    Scenario: Kundin kann ihre Rechnungsadresse eingeben
21      When die Kundin die Rechnungsadresse-Seite betritt
22      Then soll sie die Möglichkeit haben
23        * eine Anrede anzugeben
24        * den Vornamen eingeben zu können
25        * den Nachnamen eingeben zu können
26        * die Straße eingeben zu können
27        * die Hausnummer eingeben zu können
28        * 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
31      "weiter"
32      Given die Kundin hat alle Felder außer bspw. der Hausnummer
33      eingegeben
34      When sie auf "weiter" klickt
35      Then soll sie informiert werden, dass sie alle Felder ausfüllen
36      muss
37
38      Scenario: Kundin gibt invalide Daten ein
39      When die Kundin eine andere Rechnungsadresse eingibt
40      * Vorname und Nachname Sonderzeichen enthalten außer Bindestriche
41      enthält
42      * Straße Sonderzeichen außer Bindestriche und Punkte enthält
43      * Hausnummer Sonderzeichen enthält
44      * PLZ alles andere außer Zahlen enthält
45      * Stadt keine deutsche Stadt ist
46      * das @ bei der E-Mail-Adresse fehlt
47      Then soll eine Warnung angezeigt werden und der "weiter"-Button
48      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
4  nnen. Hier soll es die Möglichkeit geben, die Rechnungsadresse als
5  Lieferadresse übernehmen zu können. Alternativ sollen die Nutzer
6  die Pflichtfelder Anrede, Vornamen, Nachnamen, Straße, Hausnummer,
7  Postleitzahl, Stadt eingeben können.
8
9  Scenario: Kundin kommt auf die Lieferdaten-Seite von der
10     Rechnungsadresse-Seite aus
11     When die Kundin zum ersten Mal auf die Lieferdaten-Seite kommt
12     Then soll das Häkchen bei "Gleiche Lieferdaten wie
13     Rechnungsadresse" gesetzt sein
14     And das gleiche Formular wie von der Rechnungsadresse-Seite
15     erscheinen, mit den zuvor eingegebenen Daten
16     And das Formular soll deaktiviert sein, solange das Häkchen
17     gesetzt ist
18
19     Scenario: Kundin kommt auf die Lieferdaten-Seite von der Zahlungsdaten
20     -Seite aus
21     Given die Kundin hatte bereits zuvor die Lieferdaten ausgefüllt
```

```
13      Then sollen die zuvor eingegebenen Adressdaten weiterhin vorhanden  
14      sein  
15      Scenario: Kundin möchte die Rechnungsadresse übernehmen  
16      Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"  
17      ist gesetzt  
18      When sie auf den "weiter"-Button klickt  
19      Then soll sie auf die Seite "Zahlungsdaten" gelangen  
20      Scenario: Kundin möchte andere Lieferdaten nutzen  
21      Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"  
22      wurde entfernt  
23      When die Kundin hat alle Felder ausgefüllt  
24      And sie auf den "weiter"-Button klickt  
25      Then soll sie auf die Seite "Zahlungsdaten" gelangen  
26      Scenario: Kundin möchte andere Lieferdaten nutzen, ohne alle Felder  
27      ausgefüllt zu haben  
28      Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"  
29      wurde entfernt  
30      When die Kundin eine andere Lieferdaten eingibt  
31      * Vorname und Nachname Sonderzeichen enthalten außer Bindestriche  
32      enthält  
33      * Straße Sonderzeichen außer Bindestriche und Punkte enthält  
34      * Hausnummer. Sonderzeichen enthält  
35      * PLZ alles andere außer Zahlen enthält  
36      * Stadt keine deutsche Stadt ist  
37      * das @ bei der E-Mail-Adresse fehlt  
38      And sie auf den "weiter"-Button klickt  
39      Then soll eine Warnung angezeigt und der "weiter"-Button blockiert  
40      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  
4  Kunden die Zahlungsarten Rechnung, Lastschrift, PayPal und  
5  Kreditkarte zur Auswahl gestellt werden.  
6  
7  Scenario: Kundin kommt zum ersten Mal auf die Zahlungsdaten-Seite von  
8  der Lieferdaten-Seite  
9  When die Kundin die Seite zum ersten Mal betritt  
10 Then soll "Rechnung" vorausgewählt sein  
11  
12 Scenario: Kundin kommt auf die Zahlungsdaten-Seite von der "Bestellung  
13 abschließen"-Seite aus  
14 Given die Kundin hatte bereits zuvor die Zahlungsart ausgefüllt  
15 Then sollen die zuvor eingegebenen Zahlungsdaten weiterhin  
16 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
4   geben, bevor die Kundin die Bestellung abschließt.
5
6   Scenario: Kundin betritt die Seite
7     When die Kundin die Seite betritt soll eine Bestellübersicht über
8     die Artikel von Seite 1 angezeigt werden
9     * die Artikel angezeigt werden
10    * die Rechnungsadresse angezeigt werden
11    * die Lieferadresse angezeigt werden
12    * die Rechnungsart angezeigt werden
13    * ein "kostenpflichtig bestellen"-Button angezeigt werden
14
15   Scenario: Kundin schließt die Bestellung ab
16     When die Kundin auf den "kostenpflichtig bestellen"-Button klickt
17     Then soll eine Serverinteraktion ausgelöst werden, die die
18     Bestellung speichert
19     And die Bestellbestätigung soll dargestellt werden
```

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

Neben dem Frontend soll ein dazugehöriges Backend Teil der Demoanwendung sein. Auf Basis der Verhaltensdefinition wurde eine Architektur entworfen, welche im folgenden Abschnitt näher erläutert wird.

4.1.2 Backend

Das Backend wurde als Microservice-Architektur [NMMA16] konzipiert und orientiert sich grob an der Architektur, die der Direktversicherer derzeit betreibt. Die einzelnen Dienste wurden auf Basis von Java EE sowie Kubernetes erstellt. In Abbildung 4.1 lässt sich die Architektur betrachten, hierbei stellen Pods einzelne Containersysteme dar.

Die Kommunikation mit dem Backend erfolgt über einen Service der das „Backend4Frontend“-Pattern realisiert. Das Backend4Frontend bündelt alle Schnittstellen der dahinter liegenden Dienste, sodass das Frontend nur mit einem einzelnen Dienst kommunizieren muss. Die weiteren Dienste „Bestellungen“, „Übersetzungen“, „Addressvalidierung“ 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.

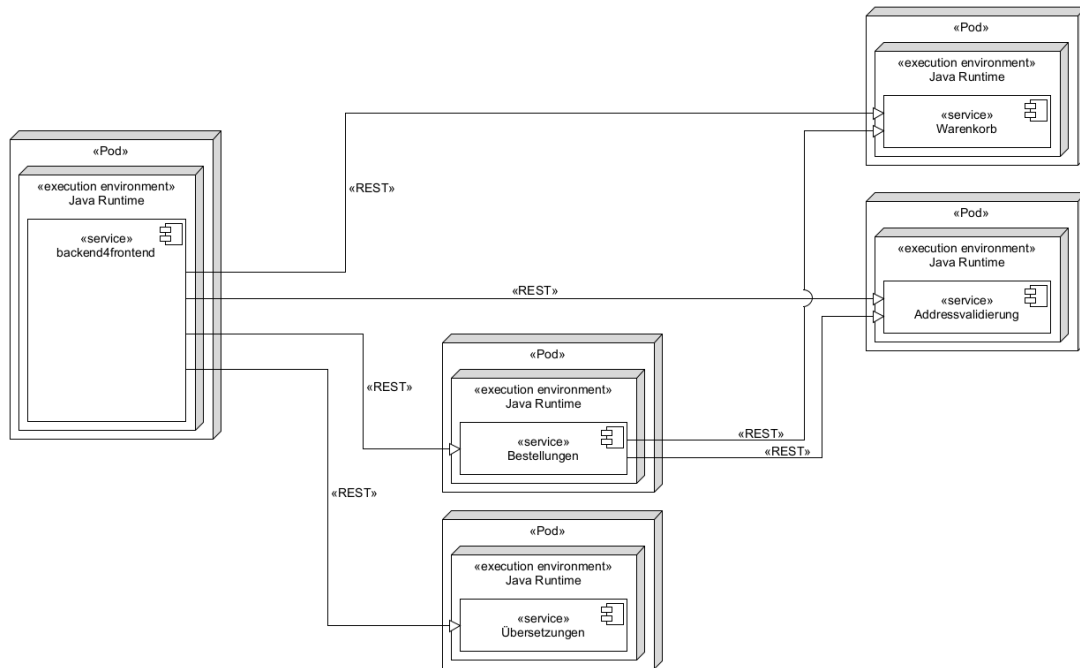


Abb. 4.1: Demoanwendung: Deployment-Diagramm. Eigene Darstellung

Die einzelnen Dienste wurden mit Eclipse MicroProfile [Ecl21] umgesetzt. MicroProfile ist eine Sammlung verschiedener Java-EE-Frameworks und Technologien zur Umsetzung von Microservices. Kern der Dienste, die auf MicroProfile basieren, sind die REST-Schnittstellen, die mit JAX-RS umgesetzt werden.

Die einzelnen Dienste wurden jeweils als Docker-Images¹ gepackt, zur Gewährleistung einer einheitlichen Umgebung auch auf unterschiedlichen Rechnern. Um ein einfaches Management der Services zu ermöglichen, wird die Plattform Kubernetes² verwendet.

4.1.3 Frontend

Auf Basis von Angular realisiert das Frontend einen Wizard, welcher mehrere aufeinander folgende Formulare in derselben SPA enthält. Anhand eines Beispieldurchlaufs durch die einzelnen Seiten wird das Frontend veranschaulicht:

Warenkorb → Rechnungsadresse → Lieferdaten → Zahlungsdaten → Bestellung abschließen → Bestellbestätigung

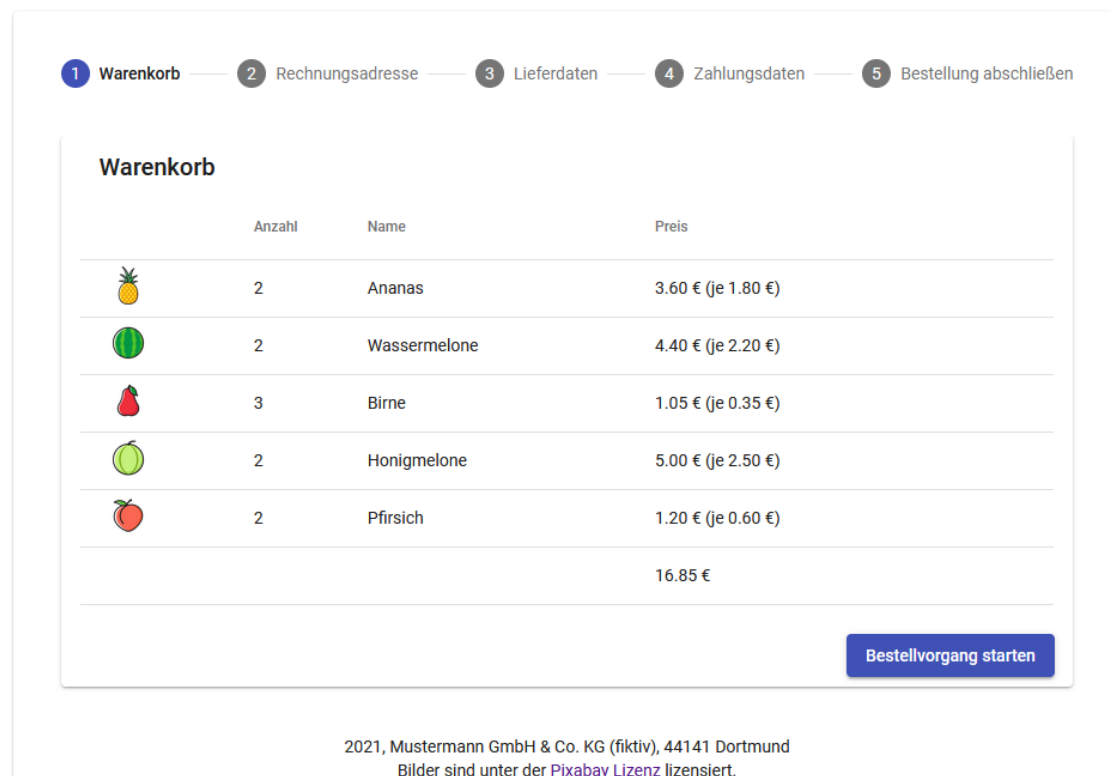
¹Docker [Doc20] ist eine Software zum Erstellen und Ausführen von Anwendungscontainern [SCF15]

²Kubernetes [Lin20] ist eine Software zum Orchestrieren Container-basierter Anwendungen [Kha17]

4.1.3.1 Warenkorb






Die erste Seite des Wizards zeigt den Inhalts des Warenkorbs eines Online-Shops für Früchte (vgl. Abbildung 4.2). 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 kein Serverabruf durchgeführt, sondern in der SPA wird ein Seitenwechsel vorgenommen.



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

Warenkorb

	Anzahl	Name	Preis
	2	Ananas	3.60 € (je 1.80 €)
	2	Wassermelone	4.40 € (je 2.20 €)
	3	Birne	1.05 € (je 0.35 €)
	2	Honigmelone	5.00 € (je 2.50 €)
	2	Pfirsich	1.20 € (je 0.60 €)
			16.85 €

[Bestellvorgang starten](#)

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

Abb. 4.2: Demoanwendung: Startseite „Warenkorb“. Eigener Screenshot

4.1.3.2 Rechnungsadresse

Auf der zweiten Seite des Wizards wird die Rechnungsadresse erfasst (vgl. Abbildung 4.3). Hierbei wird der Nutzer gebeten rechnungsrelevante Informationen, wie seine Adresse, anzugeben. 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 abgesetzt, die eingegeben Daten werden jedoch intern einer übergeordneten 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“. Eigener Screenshot

4.1.3.3 Lieferdaten

Die dritte Seite des Wizards erfasst die Lieferdaten des Nutzers. Der Nutzer kann entweder die relevanten Daten aus der Rechnungsadresse übernehmen 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 kenntlich gemacht. Anders als bei der Rechnungsadresse wird jedoch nicht der Adressvalidierungsdienst befragt, dies wird im Unterunterabschnitt 4.1.4.2 aufgefasst 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 übergeordnete Komponente innerhalb der Webanwendung weitergereicht.

Abb. 4.4: Demoanwendung: Seite „Lieferdaten“. Eigener Screenshot

4.1.3.4 Zahlungsdaten

Die vierte Seite des Wizards bittet den Nutzer zur Eingabe seiner gewünschten Zahlungsart sowie der jew. Zahlungsinformationen. Dabei kann der er zwischen 4 Zahlungsarten wählen: 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 ist die E-Mail anzugeben und bei der Auswahl der Kreditkarte folgende Kreditkarteninformationen: Karteninhaber, Kartennummer, CVC sowie das Ablaufdatum. 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 übergeordnete Komponenten übergeben.

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

Zahlungsdaten

Zahlungsart:

☐ Rechnung

☒ Lastschrift

☐ PayPal

☐ Kreditkarte

Zahlungsdaten:

Kontoinhaber
Peter Mustermann

IBAN
DE77 4405 0199 1234 5678 90

Zurück Weiter

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

Abb. 4.5: Demoanwendung: Seite „Zahlungsdaten“. Eigener Screenshot

4.1.3.5 Bestellübersicht

Die vorletzte Seite des Wizard fasst die Bestelldaten zusammen (vgl. Abbildung 4.6) und sendet die Daten bei der Bestätigung des Formulars an das Backend. 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. Zuvor eingegebene Daten sind hingegen lediglich clientseitig gespeichert und werden von einer übergreifende Komponente bereitgestellt. Weitere Eingaben des Nutzers sind auf dieser Seite 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, weiterhin erfolgt ein Seitenwechsel.

Warenkorb — Rechnungsadresse — Lieferdaten — Zahlungsdaten — 5 Bestellung abschließen

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](#)

Abb. 4.6: Demoanwendung: Seite „Bestellübersicht“. Eigener Screenshot

4.1.3.6 Bestellbestätigung

Die sechste und letzte Seite des Wizards visualisiert die Bestellbestätigung nach einer erfolgreichen Bestellung, wie in Abbildung 4.7 zu sehen ist. 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.

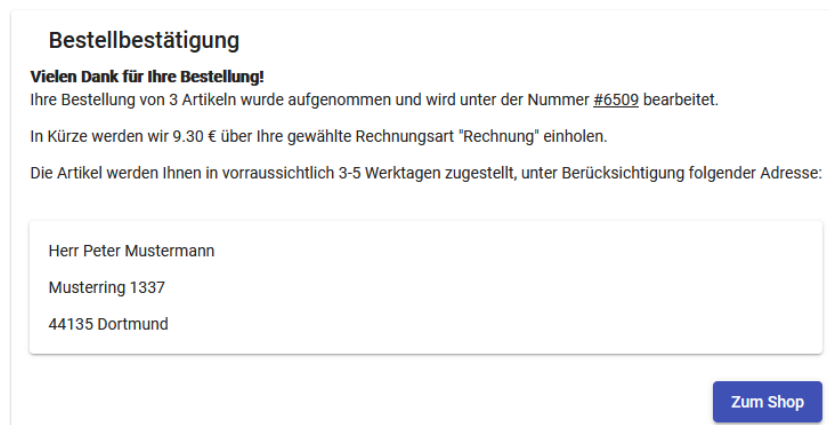


Abb. 4.7: Demoanwendung: Finale Seite „Bestellbestätigung“. Eigener Screenshot

4.1.4 Fehlerszenarien

Um später mithilfe von Observability-Werkzeugen Probleme aufzudecken und Aspekte der Nachvollziehbarkeit darzustellen, weist die Demoanwendung eine Reihe von typischen Fehlern auf, die bewusst integriert wurden.

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

4.1.4.1 „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.
Eigener Screenshot

4.1.4.2 „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

4.1.4.3 „Lange Verarbeitung“

- Problem: Beim Abrufen der Warenkorbdaten kommt es zu einer unerwünschten Wartezeit (von ca. 6-10s).

- Ursache: Dies ist eine simulierte Wartezeit im Frontend, hierbei wird eine ineffiziente Mapping-Operation nachgeahmt.

4.1.5 Repräsentation

Es 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, werden die Anforderungen näher erläutert, die das zu erstellende Proof-of-Concept erfüllen soll.

4.2 Anforderungen

Der zu erstellende Proof-of-Concept, welcher die Demoanwendung erweitern soll, hat einige Rahmenbedingungen zu erfüllen. In diesem Abschnitt werden diese Bedingungen näher beschrieben.

4.2.1 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
Basis.	Basismerkmale	Merkmale, die als selbstverständlich angesehen werden. Eine Erfüllung erhöht kaum die Zufriedenheit, jedoch eine Nichterfüllung führt zu starker Unzufriedenheit.
Leistungs.	Leistungsmerkmal	Merkmale, die der Kunde erwartet und bei nicht Vorhandensein in Unzufriedenheit äußert. Ein Vorhandensein erzeugt Zufriedenheit, beim Übertreffen umso mehr.
Begeist.	Begeisterungsmerkmal	Merkmale, die eine Herabsetzung von der Konkurrenz ermöglichen und die den Nutzenfaktor steigern. Sind sie vorhanden, steigern sie die Zufriedenheit merklich.
Unerh.	Unerhebliches Merkmal	Für den Kunden belanglos, ob vorhanden oder nicht.
Rückw.	Rückweisungsmerkmal	Diese Merkmale führen bei Vorhandensein zu Unzufriedenheit, sind jedoch beim Fehlen unerheblich.

Tab. 4.1: Merkmale nach dem Kano-Modell der Kundenzufriedenheit [Kan68]

Neben der Unterscheidung nach dem Kano-Modell werden die Anforderungen in funktionale und nicht-funktionale Anforderungen [SS97] 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.
n. f.	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 [SS97]

4.2.2 Anforderungsanalyse

Die erfassten Anforderungen stammen aus unterschiedlichen Quellen, die jeweils eigene Ziele und Wünsche für die hier angestrebte Lösung besitzen. 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

4.2.3 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	<i>Dummy</i>	<i>Begeist.</i>	<i>n. f.</i>	<i>S</i>
<i>Hier wird die Anforderung beschrieben.</i>				

Tab. 4.4: Beispiel einer Anforderung

4.2.3.1 Funktionsumfang

Id	Name	Kano-Modell	Funktionsart	Quelle
2110	Schnittstellen-Logging	Basis.	f.	S
Im Frontend ist das Aufrufen von Schnittstellen mittels einer Logmeldung zu notieren. Hierbei soll vor Aufruf geloggt werden, welche Schnittstelle aufgerufen wird und mit welchen Parametern. Nach dem Aufruf soll bei Erfolg das Ergebnisobjekt geloggt werden, und bei einem Fehler ist dieser zu notieren.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2111	Use-Case-Logging	Basis.	f.	S
Tritt im Frontend ein Use-Case auf, soll dieser im Log notiert werden. Beispielsweise soll notiert werden, wenn ein Nutzer das Absenden eines Formulars initiiert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2120	Übertragung von Logs	Basis.	f.	S
Logmeldungen des Frontends sind an ein „Log-Management“-Partnersystem weiterzuleiten. Dabei sind Logmeldungen ab dem Log-Level „DEBUG“ und höher zu übertragen.				

4 Erstellung Proof-of-Concept

Id	Name	Kano-Modell	Funktionsart	Quelle
2210	Error-Monitoring	Basis.	f.	S
Wird ein nicht abgefangener Fehler im JavaScript-Kontext geworfen, so ist dieser automatisch zu erfassen und um weitere Attribute zu ergänzen. Abgefangene und behandelte Fehler können ebenso erfasst werden, jedoch ist hierbei keine automatische Erfassung gefordert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2220	Übertragung von Fehlern	Basis.	f.	S
Sämtlich erfasste Fehler des Frontends sind an ein „Error-Monitoring“-Partnersystem weiterzuleiten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2310	Tracing	Basis.	f.	S
Im Frontend sowie im Backend sind Tracing Spans zu erstellen, die Business-Methoden sowie Schnittstellenaufrufe umschließen. Bei einem Schnittstellaufruf sind die Informationen des Spankontextes über einen Traceheader zu übergeben, sodass die subsequent erstellten Spans hiermit assoziiert werden können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2311	Tracing-Standard	Leistungs.	n. f.	A
Das Tracing soll einem gängigen Standard (wie OpenTelemetry oder OpenTracing) folgen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2320	Übertragung von Tracingdaten	Basis.	f.	S
Sämtlich erfasste Tracingdaten von Front- und Backend sind an ein „Tracing“-Partnersystem weiterzuleiten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2410	Metriken	Basis.	f.	S
Im Frontend sind beispielhaft Metriken zu erheben, wie z. B. die Anzahl an Produkten oder die Anzahl an aufgetretenen				

Id	Name	Kano-Modell	Funktionsart	Quelle
2411	Metrik-Standard	Begeist.	n. f.	A
Metriken sollen nach einem gängigen Standard (wie OpenTelemetry) erfasst werden.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2420	Übertragung von Metrikdaten	Basis.	f.	S
Sämtlich erfasste Metriken des Frontends sind an ein „Metrik“-Partnersystem weiterzuleiten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2510	Session-Replay	Basis.	f.	S
Im Frontend sind Daten zwecks Session-Replay zu erheben, welche u. A. Benutzerinteraktionen, Schnittstellenaufrufe sowie DOM-Manipulationen enthalten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2511	Schalter für Session-Replay	Basis.	f.	A
Beim ersten Aufruf des Frontends sind keine Session-Replay-Daten zu erheben. Stattdessen soll der Nutzer über einen Dialog auswählen können, ob er der Aufnahme zustimmt und erst danach sind diese Daten zu erheben.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2520	Übertragung von Session-Replay-Daten	Basis.	f.	S
Sämtliche im Frontend erfasste Daten zum Session-Replay sind an ein „Session-Replay“-Partnersystem weiterzuleiten.				

4.2.3.2 Eigenschaften

Id	Name	Kano-Modell	Funktionsart	Quelle
3010	Resilienz der Übertragung	Begeist.	f.	S
Daten, die der Nachvollziehbarkeit dienen, sollen, wenn möglich, bei einer fehlgeschlagenen Verbindung nicht verworfen werden. Sie sind (mindestens 60s) vorzuhalten und in dieser Zeit sind wiederholt (min. 3 mal) Verbindungsversuche zu unternehmen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
3020	Batchverarbeitung	Begeist.	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	Basis.	n. f.	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	Leistungs.	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.				

4.2.3.3 Partnersysteme

Id	Name	Kano-Modell	Funktionsart	Quelle
5100	Partnersystem <i>Log-Management</i>	Basis.	f.	A+S
Es existiert ein „Log-Management“-Partnersystem, zu dem Logmeldungen weitergeleitet werden und welches diese speichert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5110	Manuelle Analyse <i>Log-Management</i>	Basis.	f.	A+S
Nutzer des Systems sollen die erfassten Logmeldungen einsehen sowie diese filtern können. Die Filterung erfolgt auf Basis der Eigenschaften der Logmeldung (bspw. des Log-Levels).				

Id	Name	Kano-Modell	Funktionsart	Quelle
5200	Partnersystem <i>Error-Monitoring</i>	Basis.	f.	A+S
Es existiert ein „Error-Monitoring“-Partnersystem, zu dem Fehler weitergeleitet werden und welches diese persistiert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5210	Manuelle Analyse <i>Error-Monitoring</i>	Basis.	f.	A+S
Nutzer des Systems sollen die erfassten Fehler einsehen sowie diese filtern können. Die Filterung erfolgt auf Basis der Eigenschaften der Fehler (bspw. der Fehlername).				

4 Erstellung Proof-of-Concept

Id	Name	Kano-Modell	Funktionsart	Quelle
5220	Visualisierung <i>Error-Monitoring</i>	Leistungs.	f.	S
Die Fehler sollen bspw. in Histogrammen grafisch dargestellt werden können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5230	Alerting <i>Error-Monitoring</i>	Begeist.	f.	A+S
Bei Auftreten von bestimmten Fehlern oder einer Anzahl von Fehlern soll eine Meldung erzeugt werden können (per E-Mail, Slack, o. Ä.).				

Id	Name	Kano-Modell	Funktionsart	Quelle
5300	Partnersystem <i>Tracing</i>	Basis.	f.	A+S
Es existiert ein „Tracing“-Partnersystem, welches die Tracingdaten konsumiert und speichert. Zusammengehörige Spans sind zu Traces zusammenzufassen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5310	Manuelle Analyse <i>Tracing</i>	Basis.	f.	A+S
Die erfassten Tracingdaten sind für die Nutzer des Systems einsehbar und können gefiltert werden. Die Filtierung erfolgt auf Basis von Eigenschaften der Tracingdaten (wie Name des meldenden Systems).				

Id	Name	Kano-Modell	Funktionsart	Quelle
5320	Visualisierung <i>Tracing</i>	Basis.	f.	A+S
Das Partnersystem, zu dem die Tracingdaten weitergeleitet werden, soll diese grafisch als Trace-Gantt-Diagramm darstellen können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5400	Partnersystem <i>Metriken</i>	Leistungs.	f.	A+S
Es existiert ein „Metrik“-Partnersystem, zu dem Metriken weitergeleitet werden und welches diese persistiert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5410	Visualisierung <i>Metriken</i>	Leistungs.	f.	A+S
Metriken sind grafisch darstellbar, bspw. in Histogrammen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5420	Alerting <i>Metriken</i>	Begeist.	f.	S
Bei Auftreten von bestimmten Metrikwerten oder Überschreitungen von Schwellen soll eine Meldung erzeugt werden können (per E-Mail, Slack, o. Ä.).				

Id	Name	Kano-Modell	Funktionsart	Quelle
5500	Partnersystem <i>Session-Replay</i>	Basis.	f.	A+S
Es existiert ein „Session-Replay“-Partnersystem, zu dem die Daten zur Session-Replays gesendet werden und welches diese analysiert und speichert.				

Id	Name	Kano-Modell	Funktionsart	Quelle
5510	Nachstellung <i>Session-Replay</i>	Basis.	f.	A+S
Dieses System soll anhand der Daten jede aufgezeichnete Benutzersitzung in Videoform nachstellen.				

4.3 Konzept

Anhand der Anforderungen und den zuvor in Abschnitt 3.2 recherchierten Technologien, gilt es nun ein Konzept zu erstellen, wie die Demoanwendung zu erweitern ist. Dabei ist das Grundziel, die Nachvollziehbarkeit dieser Anwendung zu erhöhen. In den Anforderungen werden vier Arten von Daten genannt, die es zu erheben und zu nutzen gilt. Darunter die drei „Grundpfeiler der Observability“ Logs, Metriken und Traces sowie die gesondert zu betrachtende Methodik des Session-Replays.

In Unterabschnitt 3.2.6 wurden kriteriengeleitet bereits Technologien identifiziert, mit denen eine verbesserte Nachvollziehbarkeit erreicht werden kann: Splunk, Sentry, Jaeger sowie LogRocket (vgl. Tabelle 4.5). Bei dieser Übersicht ist zu betrachten, dass Splunk die grundsätzlichen Funktionalitäten von Sentry abdecken kann. Andere Werkzeuge des Error-Monitorings weisen die gleiche Überschneidung auf. Lediglich das Issue-Management, welches in diesen Technologien oft ein fester Bestandteil ist, kann nicht allein mit Splunk abgebildet werden - diese Funktionalität ist aber für die in dieser Arbeit verfolgten Ziele nicht relevant. Weiterhin gehören Bug-Tracking-Systeme bereits unabdingbar zu Softwareprojekten [SC05], ein weiteres ähnlich agierendes System könnte hierbei eine Dopplung darstellen und somit unerwünscht sein. Aus diesem Grund wird auf Sentry verzichtet, dies geht zudem mit der Anforderung 3100 einher, welche eine geringe Anzahl an zusätzlichen Partnersystemen vorsieht.

Technologie	IM	ASM	RUM	Error-Monitoring	Log-Mgmt.	Tracing	Session-Replay
Splunk	ja(*)	ja(*)	ja(*)	ja(*)	ja		
Jaeger						ja	
Sentry			ja(*)	ja			
LogRocket			ja	ja			ja

Tab. 4.5: Übersicht der ausgewählten Technologien

Eine weitere Reduzierung der Anzahl der hinzukommenden Technologien konnte jedoch nicht erreicht werden. Splunk, Jaeger und LogRocket besitzen jeweils Kernfunktionalitäten, die nicht durch andere Werkzeuge abbildbar sind. Splunk ist allen voran ein Log-Management-System, welches durch die Anforderungen erwünscht ist und dabei können die anderen Technologien es nicht ersetzen. Jaeger besitzt mit Distributed-Tracing und LogRocket mit dem Session-Replay in Videoform ebenso Kernfunktionalitäten, die nicht mit anderen Werkzeugen abbildbar sind. Aus diesem Grund soll die Erweiterung auf Basis dieser 3 Technologien erfolgen.

Genauer sollen, wie in Abbildung 4.9 visualisiert, aus dem Frontend und den Backend-Diensten Traces erhoben werden und an Jaeger übermittelt werden. Darüber hinaus sind im Frontend Logs, Fehler und Metriken zu erheben und an Splunk zu senden, dabei sind nach Anforderungen Anforderung 5220 und Anforderung 5410 Fehler und Metriken visuell aufzubereiten.

Zudem sollen über die von LogRocket bereitgestellte JavaScript-Bibliothek die Daten zum Session-Replay erfasst werden und an LogRocket gemeldet werden, jedoch nur wenn der Nutzer zuvor zustimmt (vgl. Anforderung 2511). Splunk und Jaeger sollen zudem lokal, also OnPremise, aufgesetzt werden. LogRocket hingegen bietet dies nur für Unternehmenskunden an und wird somit als SaaS-Produkt zum Einsatz kommen.

Anhand dieses Konzeptes erfolgt im nächsten Abschnitt die Umsetzung dessen. Genauer wird die Implementierung der Erweiterung des Back- und Frontends näher erläutert.

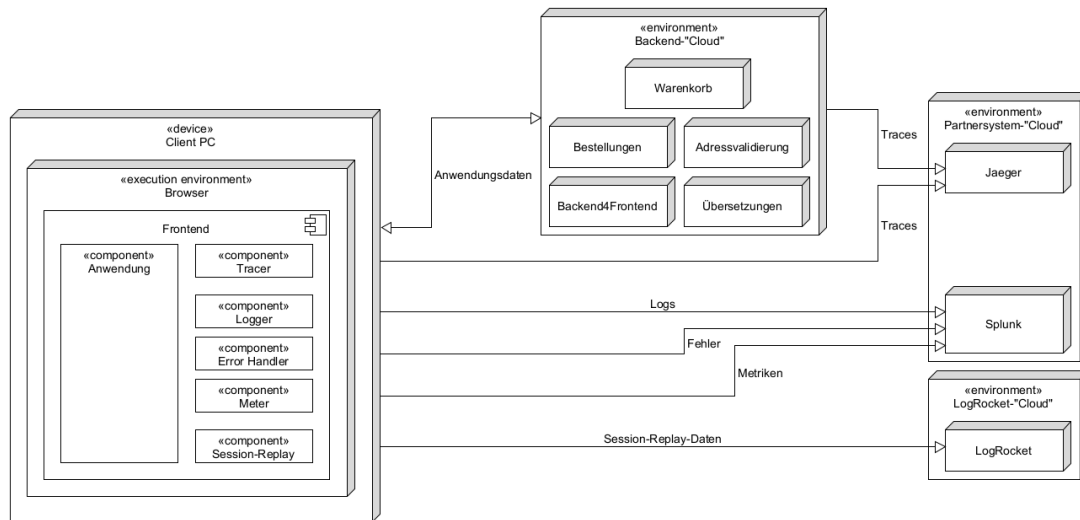


Abb. 4.9: Architektur des Konzeptes. Eigene Darstellung

4.4 Implementierung

4.4.1 Backend

Die Dienste wurden 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 [Jae21a] 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 wurden diese relevanten Methoden annotiert 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      /* Berechnung zur Bestellung */
14
15      return new Receipt(/* Belegdaten */);
16  }
17 }

```

Quellcode 4.6: Beispielhafter Einsatz der @Traced-Annotation

In Jaeger erzeugt der o. g. Quellcode die in Abbildung 4.10 zu sehenden Spans. Bei diesem Ausschnitt ist erkennbar, dass der Dienst „Bestellungen“ für eine ursprüngliche Anfrage zwei Anfragen an den Dienst „Adressvalidierung“ sendet, jeweils für die Rechnungs- und Lieferadresse. Neben den Traces werden keine weiteren Daten von Backend-Komponenten erhoben, da das Hauptaugenmerk der Arbeit auf dem Frontend liegt.

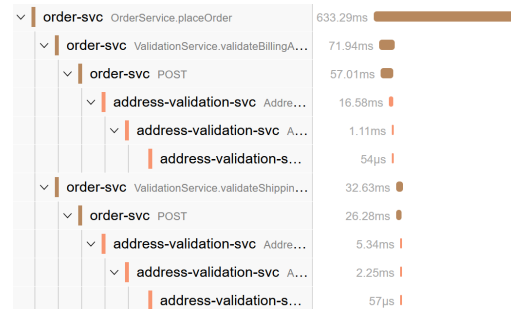


Abb. 4.10: Ausschnitt des Traces zu Quellcode 4.6

4.4.2 Frontend

4.4.2.1 Datenweiterleitung im „Backend4Frontend“

Da Logs, Metriken und Fehler mit Splunk gesammelt werden, aber Splunk keine direkt aus dem Browser ansprechbare Schnittstelle bietet, müssen diese über einen Dienst an Splunk weitergeleitet werden (vgl. Unterunterabschnitt 3.2.7.1). Neben diesen Daten sind zudem Traces auf dem Frontend an Jaeger zu berichten, dies ist aber erneut nicht möglich, da kein browserkompatibler Jaeger-Client identifiziert werden konnte. Eine nähere Betrachtung dessen erfolgt im nächsten Abschnitt.

Da der Dienst „Backend4Frontend“ bereits zum Weiterleiten verwendet wird, wurde dieser erweitert, sodass er Traces, Logs, Metriken sowie Fehler entgegennehmen kann und diese an Jaeger bzw. Splunk sendet. Vor der eigentlichen Weiterleitung werden die Daten mit Kontextinformationen angereichert, wie bspw. der User-IP oder der Browserversion. Die Traces werden dann über eine OpenTelemetry-Anbindung an Jaeger gesendet, die anderen Daten werden an die HEC-Schnittstelle [Spl21a] von Splunk übertragen.

4.4.2.2 Traces und Metriken

Das Frontend erhebt ebenso wie das Backend Traces, zusätzlich werden aber auch Metriken, Logmeldungen und Fehler erfasst und gemeldet. Traces und Metriken werden auf Basis von OpenTelemetry-JavaScript-Komponenten [Ope21b] erhoben. Diese Komponenten werden in einem eigenen Angular-Modul (siehe Quellcode 4.7) initialisiert und der 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 folgenden 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. Wie der Span weiterverwendet wird, ist im nächsten Absatz beschrieben.

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

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 [Jae21a] 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     // Starte einen neuen Span mit Elternreferenz
15     const span = this.traceUtil.startChildSpan(
16       this.tracer, 'LocalizationService.getTranslations', parentSpan,
17       { 'shoppingCartId': window.customer.shoppingCartId }
18     );
19
20     // Generiere aus OTel span einen Jaeger-kompatiblen HTTP-Header
21     const jaegerTraceHeader = this.traceUtil.
      serializeSpanContextToJaegerHeader(span.context());
22
23     // Erhöhe die "requestCounter"-Metric
24     this.requestCounter.add(1, { 'component': 'LocalizationService' });
25
26     return this.http.get<Localization>(
27       this.localizationServiceUrl,
28       { headers: { 'uber-trace-id': jaegerTraceHeader } }
29     )
30     .pipe(
31       tap(
32         (val) => {
33           this.log.info('getTranslations(): returnVal = ', val);
34
35           span.end();
36         },
37         (err) => {
38           // Protokolliere Fehler
39           span.recordException({ code: err.status, name: err.name,
      message: err.message });
40           span.end();
41         }
42       ),
43     );
44   }
45 }

```

Quellcode 4.9: Service zum Abrufen der Übersetzungsdaten

Um das Erheben von Traces und Metriken im Frontend zu realisieren, wurden OpenTelemetry-Komponenten herangezogen, da wie in Unterabschnitt 3.1.3 beschrieben, OpenTelemetry einen vielversprechenden Standard darstellt. Weiterhin konnte keine Bibliothek identifiziert werden, die die Traces erhebt und direkt an Jaeger sendet. Es gibt zwar einen Jaeger-Client für Node.js³, jedoch befindet sich das browserkompatible Pendant⁴ seit 2017 in den Startlöchern [Lof17]. Zudem existiert ein OTel-Exporter für Jaeger⁵, welcher jedoch auch nur mit Node.js funktioniert. Grund hierfür ist, dass das zugrundeliegende Protokoll gRPC [Lin21] nicht komplett aus einer Browserumgebung aus unterstützt wird [Bra19].

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 bereits im Frontend in ein Splunk-kompatibles Logformat konvertiert. Nach der Konvertierung werden die Daten an den `SplunkForwardingService` übergeben, welcher in Unterunterabschnitt 4.4.2.5 näher beschrieben wird.

4.4.2.3 Logging

Das Logging im Frontend wurde über das npm [npm21] Paket `ngx-logger`⁶ realisiert, welches eine speziell an Angular angepasste Logging-Lösung darstellt. Da dieses Paket extra an Angular angepasst ist, lässt es sich ohne großen Aufwand als Modul einbinden und einsetzen (vgl. Quellcode 4.10).

```
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`

³Jaeger-Client für Node.js: <https://github.com/jaegertracing/jaeger-client-node>

⁴Jaeger-Client für Browser: <https://github.com/jaegertracing/jaeger-client-javascript/>

⁵OTel Jaeger Exporter: <https://github.com/open-telemetry/opentelemetry-js/tree/main/packages/opentelemetry-exporter-jaeger>

⁶`ngx-logger` auf GitHub: <https://github.com/dbfannin/ngx-logger>

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. Die Funktionsweise der Weiterleitung wird an späterer Stelle genauer beschrieben.

```
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

4.4.2.4 Fehler

Die `ErrorHandler`-Hook von Angular übermittelt aufgetretene und unbehandelte Fehler an den `SplunkForwardingErrorHandler`. Weiterhin ist der `ErrorHandler` ein `Injectable`, wodurch er in andere Angular-Klassen „injected“ werden kann. Der `ErrorHandler` wird bspw. bei den Schnittstellen dazu verwendet, um dort bereits behandelte Fehler auch an Splunk zu übermitteln.

Wird ein Fehler gemeldet, werden zunächst die Fehlerinformationen in einen Splunkdatensatz konvertiert und dann über den **SplunkForwardingService** an das „Backend4Frontend“ 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                 // vgl. 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

4.4.2.5 Weiterleitung an Splunk

Logs, Metriken und Fehler über den **SplunkForwardingService** an das „Backend4Frontend“ gesendet und letztendlich an Splunk übermittelt. Damit nicht jedes Datenobjekt einen Aufruf auslöst, gruppiert dieser **Service** die eintreffenden Daten.

Im Quellcode 4.13 ist erkennbar, dass alle 5s geprüft wird, ob Daten zur Weiterleitung zur Verfügung stehen. Ebenso ist zu sehen, dass beim Fehlschlag des Sendens eines Batches,

fünf Wiederholversuche gestartet werden, mit einer Wartezeit von 15s. Die grundlegende Funktionalität dessen wurde mit RxJS [Goo21c] umgesetzt, einer Kernbibliothek von Angular, die Reactive Programming [SF16] in JavaScript erlaubt. Weiterhin werden die Daten an Splunk nicht im Textformat übertragen, sondern direkt in JSON. Die JSON-Objekte werden automatisch von Splunk interpretiert und die enthaltenen Felder zugreifbar gemacht. Durch das gewählte Format [TVNP13] ist auch die Anforderung 3200 zum Structured-Loggings erfüllt. Ausgesuchte Logmeldungen und Fehler sind in Quellcode 4.14 zu betrachten.

```
1 export class SplunkForwardingService {
2   private batchQueue: SplunkEntry[] = [];
3
4   constructor(private http: HttpClient) {
5     // Prüfe alle 5s ob Daten zum Weiterleiten existieren
6     interval(5000)
7       .subscribe(() => {
8         const batch = this.batchQueue;
9
10        this.batchQueue = [];
11
12        this.sendBatch(batch)
13          .pipe(
14            // 5 Versuche mit einer Wartezeit von jeweils 15s
15            retryWhen(errors => errors.pipe(delay(15000), take(5)))
16          )
17          .subscribe();
18      });
19  }
20
21  public forwardEvents(entries: SplunkEntry[]): void {
22    for(const entry of entries) {
23      // Füge Kontextinformationen hinzu
24      entry.event.path = window.location.href;
25      entry.event.shoppingCartId = window.customer.shoppingCartId;
26
27      this.batchQueue.push(entry);
28    }
29  }
30
31  private sendBatch(batch: SplunkEntry[]): Observable<void> {
32    return this.http.post<void>(this.logBatchServiceUrl, batch);
33  }
34 }
```

Quellcode 4.13: Implementierung des SplunkForwardingService


```

1  [
2      {
3          "event": {
4              "message": "submit()",
5              "path": "http://localhost:4200/checkout",
6              "severity": 2,
7              "shoppingCartId": "f7db5e54-ec9a-9161-ff27-2468b1f41f54",
8              "timestamp": "2021-03-05T17:56:54.548Z"
9          },
10         "sourcetype": "log"
11     }, {
12         "event": {
13             "message": "order(): placing order, data = ",
14             "param0": "{\\"shoppingCartId\\":\\"f7db5e54-ec9a-9161-ff27-2468b1f41f54\\",...}",
15             "path": "http://localhost:4200/checkout",
16             "severity": 2,
17             "shoppingCartId": "f7db5e54-ec9a-9161-ff27-2468b1f41f54",
18             "timestamp": "2021-03-05T17:56:54.553Z"
19         },
20         "sourcetype": "log"
21     }, {
22         "event": {
23             "component": "OrderService",
24             "description": "Anfrage zum Bestelldienst war nicht erfolgreich, Fehler = HTTP 400 Bad Request",
25             "environment": "development",
26             "frontendModel": {
27             },
28             "message": "Http failure response for http://localhost:53246/data/order: 502 Bad Gateway",
29             "name": "HttpErrorResponse",
30             "path": "http://localhost:4200/checkout",
31             "shoppingCartId": "f7db5e54-ec9a-9161-ff27-2468b1f41f54"
32         },
33         "sourcetype": "error"
34     }, {
35         "event": {
36             "logRocketSessionURL": "n/a",
37             "param0": "{\\"name\\":\\"HttpErrorResponse\\",\\"message\\":...}",
38             "path": "http://localhost:4200/checkout",
39             "severity": 4,
40             "shoppingCartId": "f7db5e54-ec9a-9161-ff27-2468b1f41f54",
41             "timestamp": "2021-03-05T17:56:54.765Z"
42         },
43         "sourcetype": "log"
44     },
45 ]

```

Quellcode 4.14: Beispiel eines Batches

4.4.2.6 Session-Replay

Die Übermittlung der Daten an LogRocket wird erst aktiviert, wenn der Nutzer explizit der Aufzeichnung zustimmt. Dies bedeutet jedoch auch, dass bis zur Zustimmung keine Sitzungsdaten aufgenommen wurden. Klickt der Nutzer jedoch auf den, unten rechts in der Anwendung schwebenden, Button (vgl. Abbildung 4.11), wird ein Einverständnis-Dialog geöffnet. Hierbei (vgl. Abbildung 4.12) erhält der Nutzer eine Übersicht welche Daten aufgenommen werden und an wen diese dann weitergesendet werden. Stimmt der Nutzer zu, wird LogRocket initialisiert, wie im Quellcode 4.15 sowie Quellcode 4.16 zu sehen ist. 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.

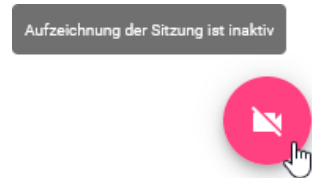


Abb. 4.11: Button zum Öffnen des Dialogs

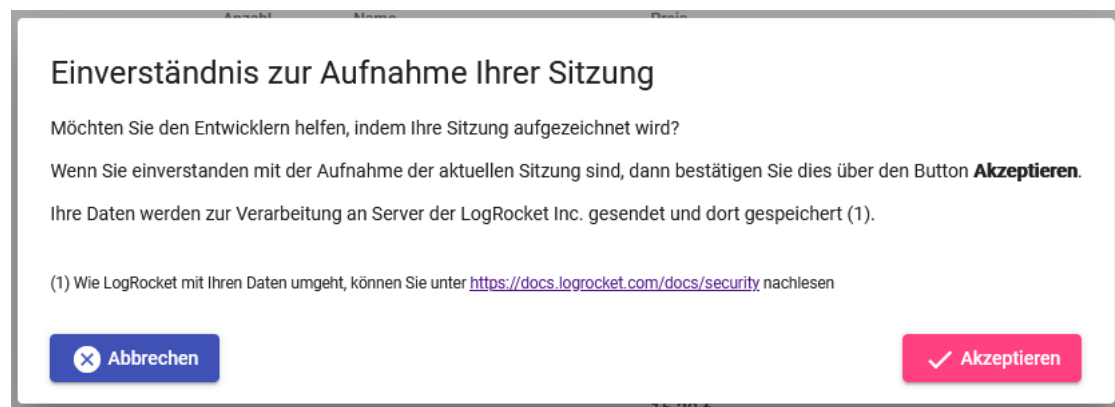


Abb. 4.12: Einverständnis-Dialog

```

1 <h1 mat-dialog-title>Einverständnis zur Aufnahme Ihrer Sitzung</h1>
2 <div mat-dialog-content>
3   <!-- Textinhalt -->
4 </div>
5 <div mat-dialog-actions class="cardActions">
6   <button mat-raised-button (click)="closeDialog()" color="primary">
7     Abbrechen
8   </button>
9   <button mat-raised-button (click)="activateLogRocket()" color="accent">
10    Akzeptieren
11  </button>
12 </div>

```

Quellcode 4.15: Angular HTML-Template der Einverständnis-Komponente

```

1 export class RecordConsentDialogComponent {
2   constructor(
3     public dialogRef: MatDialogRef<RecordConsentDialogComponent>,
4     @Inject(APP_CONFIG) private config: AppConfig
5   ) { }
6
7   closeDialog() {
8     this.dialogRef.close(window.logrocketData.sessionURL);
9   }
10
11  activateLogRocket() {
12    // Initialisiere die Aufnahme mit LogRocket
13    LogRocket.init(this.config.logRocketAppId, {});
14
15    // Starte eine neue Sitzung
16    LogRocket.startNewSession();
17
18    // Stelle die LogRocket sessionURL bereit
19    LogRocket.getSessionURL((sessionURL) => {
20      window.logrocketData.sessionURL = sessionURL;
21
22      this.dialogRef.close(sessionURL);
23    });
24  }
25 }

```

Quellcode 4.16: Initialisierung von LogRocket in der Einverständnis-Komponente

4.4.3 Architektur

Abschließend lässt sich somit folgende, in Abbildung 4.13 zu betrachtende, Architektur visualisieren. Bei ① lässt sich die Übertragung von Spans an Jaeger betrachten, bei dem die Backend-Dienste aufgrund der verwendeten Java-Anbindung (`io.jaegertracing:jaeger-client`) die Spans über das Protokoll „Apache Thrift“ [Apa21] berichten und das „Backend4Frontend“ verwendet zur Weiterleitung eine OTel-Anbindung an Jaeger, die wiederum gRPC verwendet. Die Übertragung von Log-, Metrik sowie Fehlerdaten lässt sich bei ② betrachten. Dabei handelt es sich um Daten, die zuvor vom Frontend gesendet wurden (vgl. ③). Die Kommunikation mit LogRocket (vgl. ④) verläuft ohne Weiterleitung und mit einem proprietärer Protokoll auf Basis von gRPC [Log21b].

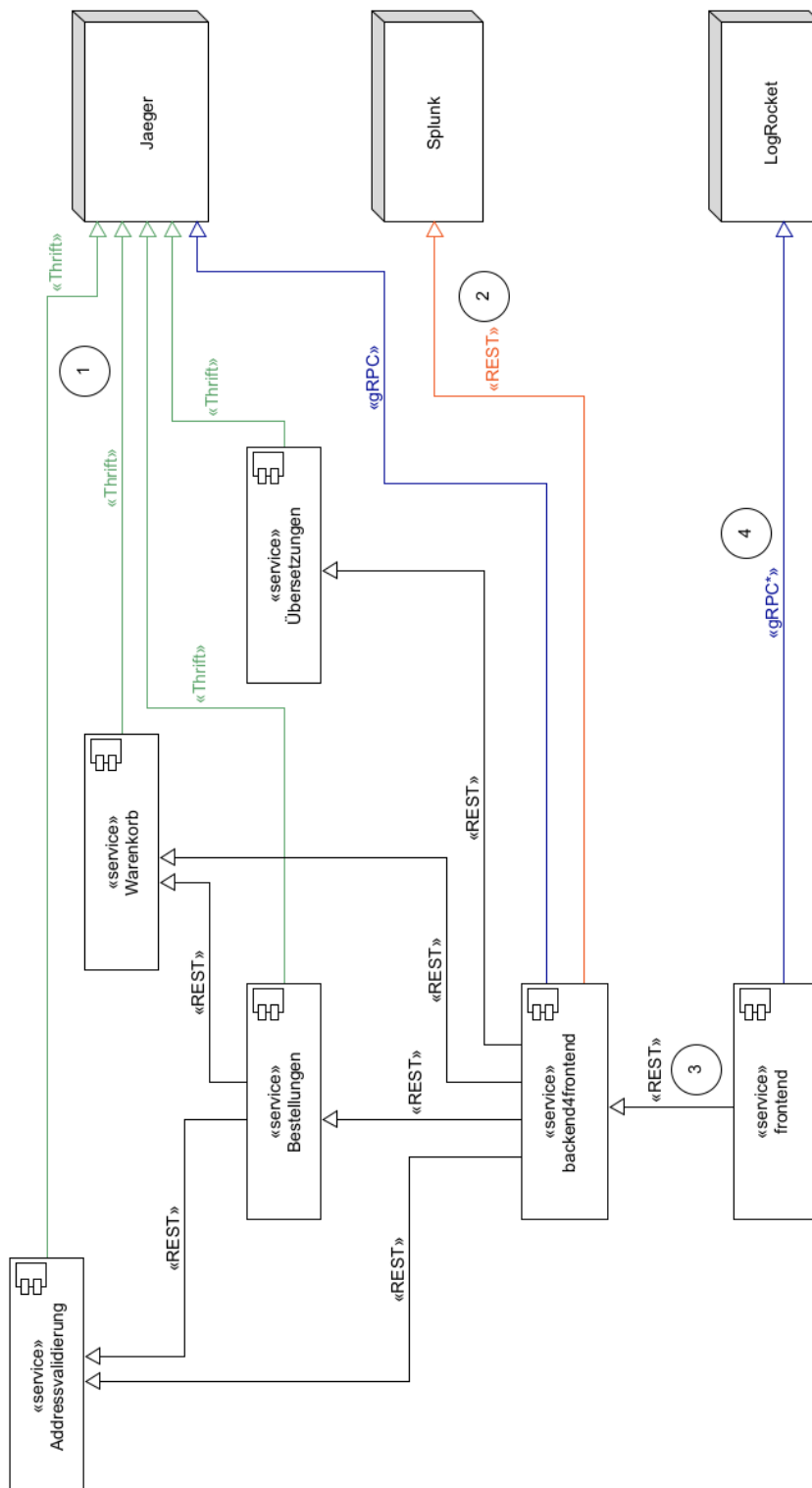


Abb. 4.13: Architektur des Proof-of-Concepts. Eigene Darstellung

5 Ergebnis

5.1 Demonstration

Der Nutzen des Konzeptes und dessen Umsetzung wird nachfolgend beispielhaft vorgestellt. Auf Basis der eingebauten Fehlerszenarien in der Demoanwendung soll der Mehrwert der erstellten Lösung evaluiert werden. Folgend wird näher beleuchtet, wie die drei Fehlerszenarien aufgedeckt werden können.

5.1.1 Aufdecken des Szenarios „Keine Übersetzungen“

Im Fehlerszenario „Keine Übersetzungen“ werden in der Webanwendung die Übersetzungsschlüssel, statt der tatsächlichen Produktnamen, angezeigt (vgl. Unterunterabschnitt 4.1.4.1). Da es sich hierbei um einen Fallback handelt, wurde in der Webanwendung darauf verzichtet einen Fehler zu werfen. Dieser würde in Splunk hervorgehoben werden. Jedoch lassen sich in Splunk die Logdaten durchsuchen und so Sitzungen herausfinden, bei denen dieses Problem aufgetreten ist (vgl. Abbildung 5.1). Darauf basierend konnten verwandte Logs derselben „Sitzung“ inspiziert werden, alleine anhand dessen konnte jedoch nicht das Fehlverhalten nachvollzogen werden. Aber die gefundene `shoppingCartId` bietet die Möglichkeit, damit in weiteren Systemen nachzuforschen.

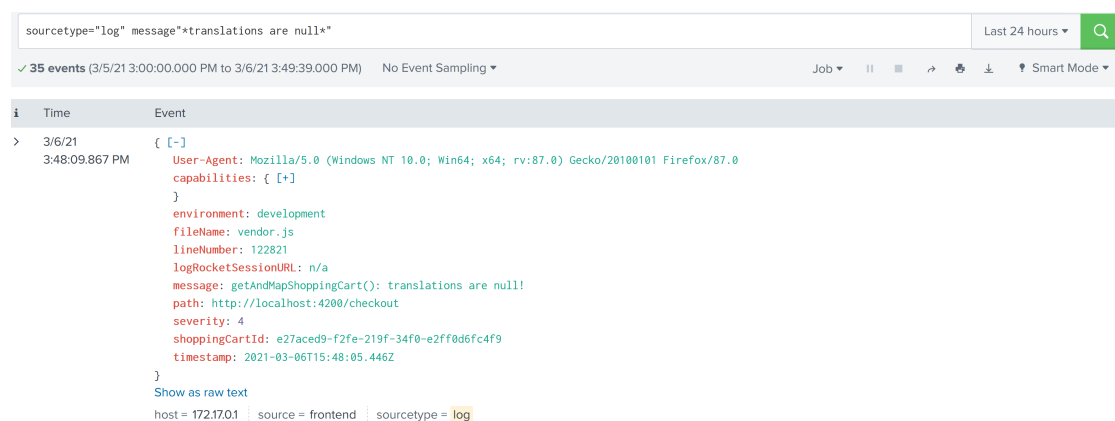


Abb. 5.1: Suche nach Logs zu fehlenden Übersetzungen in Splunk

In Jaeger lassen sich anhand der `shopping-CartId` Traces finden (vgl. Abbildung 5.2). Einer dieser Traces ist mit einem Fehler markiert und wurde ursprünglich durch die Funktion `getAndMapShoppingCart` im Frontend ausgelöst. Dabei handelt es sich um eine Funktion, die die Warenkorb- sowie Übersetzungsdaten abrufen und zusammengeführt.

Bei Klick auf den Trace wird das entsprechende Trace-Gantt-Diagramm geöffnet (vgl. Abbildung 5.3). Beim dargestellten Aufruf ist im Übersetzungsdienst ein Fehler aufgetreten, welcher vermutlich für die fehlenden Übersetzungen verantwortlich ist. Im Span ist zudem ein Log hinterlegt, welche einen Hinweis liefert: `configuration is invalid`. Weiterhin ist auch der genaue Kubernetes-Pod (`localization-svc-003`) zu sehen, der die Abfrage durchführte. Somit sind dem problemlösenden Entwickler viele notwendige Informationen geboten, die ihm dabei helfen das Problem zu lösen.

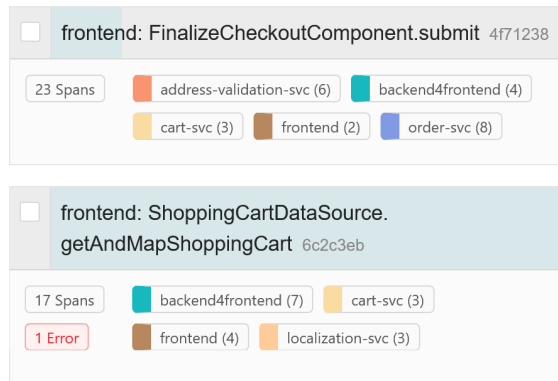


Abb. 5.2: Suchergebnisse in Jaeger zu spezieller `shoppingCartId`

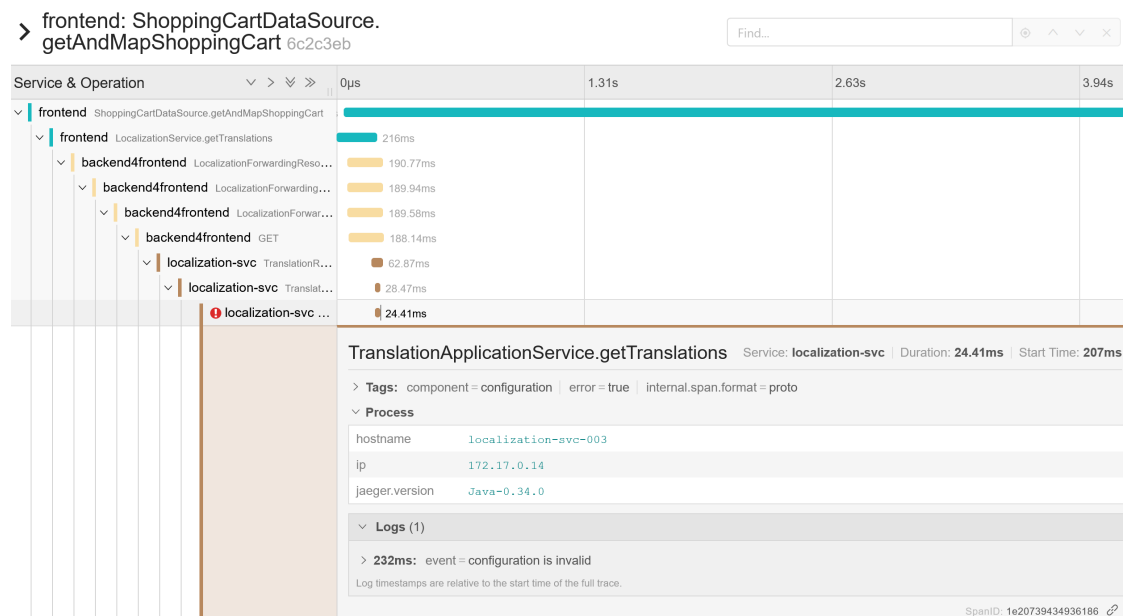


Abb. 5.3: Ausschnitt des Traces zur Übersetzungsanfrage in Jaeger

5.1.2 Aufdecken des Szenarios „Ungültige Adressen sind gültig“

Das Backend bietet angebundene Systemen die Möglichkeit, eine Adresse zu validieren. Bei der Rechnungs- sowie der Lieferadresse sollte dies auch der Fall sein. Jedoch kommt es dazu, dass Nutzer ungültige Adressen angeben können und dies beim Absenden einer Bestellung zu einem Fehler führt (vgl. Unterunterabschnitt 4.1.4.2). Durch die Suche in Splunk nach Logmeldungen der Angular-Komponente `FinalizeCheckoutComponent` findet sich eine Fehlermeldung und eine entsprechende `shoppingCartId`.

Mit Ausnahme der Fehlermeldung liefert Splunk keine sprechenden Informationen, so dass erneut auf Jaeger zurückgegriffen wird. In Jaeger findet sich ein entsprechender Trace (vgl. Abbildung 5.4), der über die Situation Aufschluss gibt. Dabei ist zu erkennen, dass im Bestelldienst beide Adressen erneut validiert werden und die Anfrage bzgl. der Lieferadresse fehlschlägt. Auf Basis dieser Informationen kann der Entwickler erkennen, dass in der SPA keine Validierung für die Lieferadresse durchgeführt wird.

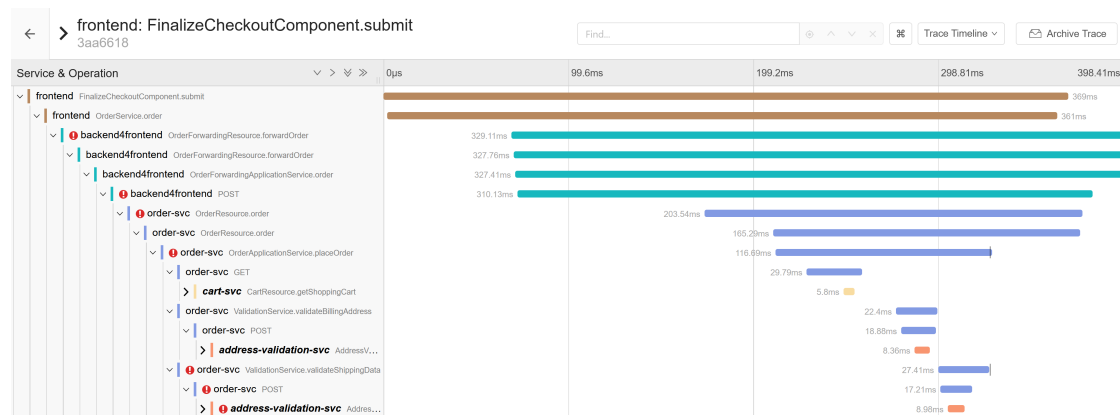


Abb. 5.4: Ausschnitt des Traces zur Bestellanfrage in Jaeger

5.1.3 Aufdecken des Szenarios „Lange Verarbeitung“

In diesem Szenario kommt es bei der Anzeige des Warenkorbes zu einer zeitlichen Verzögerung, sodass einige Sekunden vergehen, bis dieser sichtbar ist (vgl. Unterunterabschnitt 4.1.4.3). In diesem Fall ist Splunk keine große Hilfe, da eine zeitliche Abfolge alleine durch Logs schwer nachvollziehbar ist. Durch LogRocket ist das resultierende Verhalten gut nachzuvollziehen. Ein Video zu diesem Fehlerszenario ist im Anhang zu finden (siehe Unterabschnitt 7.2.1).

Auch in dieser Situation ist Jaeger ein wesentlicher Faktor. Anhand der Traces lassen sich nicht nur hierarchische Beziehungen nachvollziehen, sondern auch zeitliche. Eine Trace-Gantt-Visualisierung eines Traces, bei dem der Warenkorb abgerufen wird, ist in Abbil-

dung 5.5 zu betrachten. Dabei sticht hervor, dass der überwiegende Anteil der Zeit nicht in den Backendanfragen verloren geht, sondern durch die Operation `mapShoppingCart` im Frontend. Mithilfe dieser Informationen kann ein Entwickler nun hergehen und diese Operation verbessern.

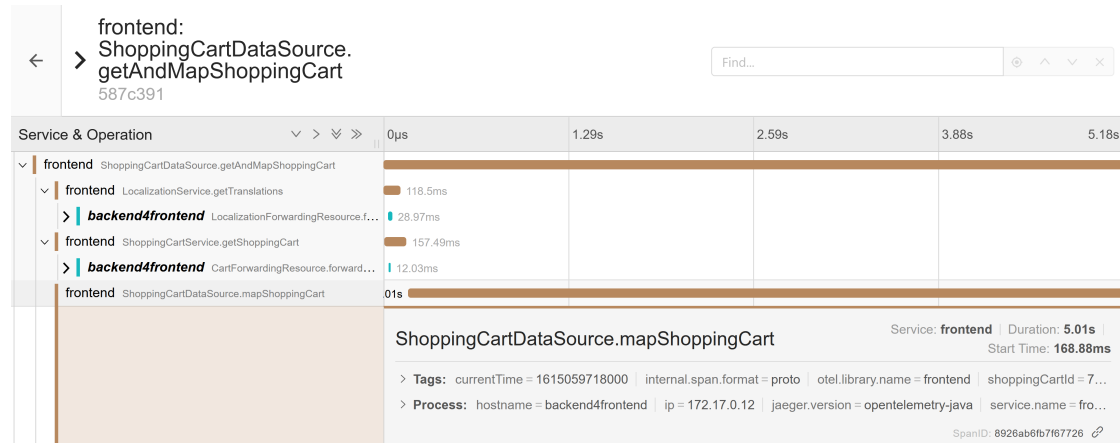


Abb. 5.5: Ausschnitt des Traces zur Warenkorbanfrage in Jaeger

5.2 Bewertung der Anforderungen

Vor der Erstellung des Konzeptes und der eigentlichen Implementierung wurden dafür Anforderungen definiert. Diese sollen nun überprüft werden, ob und in welchem Umfang sie umgesetzt wurden. Dafür werden die Anforderungen tabellarisch und in kurzer Form dargestellt. Um den Grad der Erfüllung zu beschreiben, existiert zudem die Spalte *Erfüllungsgrad*. Dabei handelt es sich um einen Prozentwert von 0%–100%, wobei 0% keine Umsetzung der Anforderung bedeutet und 100% eine vollständige. Werte, die dazwischen liegen, werden nachgehend genauer erläutert, da hierfür keine allgemeine Beschreibung gewählt werden kann.

Id	Titel	Kano-Modell	Funktionsart	Quelle	Erfüllungsgrad
Funktionsumfang					
2110	Schnittstellen-Logging	Basis.	f.	S	100%
2111	Use-Case-Logging	Basis.	f.	S	100%
2120	Übertragung von Logs	Basis.	f.	S	100%
2210	Error-Monitoring	Basis.	f.	S	100%
2220	Übertragung von Fehlern	Basis.	f.	S	100%
2310	Tracing	Basis.	f.	S	100%

Id	Titel	Kano-Modell	Funktions-art	Quelle	Erfüllungs-grad
2311	Tracing-Standard	Leistungs.	n. f.	A	100%
2320	Übertragung von Tracingdaten	Basis.	f.	S	100%
2410	Metriken	Basis.	f.	S	100%
2411	Metrik-Standard	Begeist.	n. f.	A	100%
2420	Übertragung von Metrikdaten	Basis.	f.	S	100%
2510	Session-Replay	Basis.	f.	S	100%
2511	Schalter für Session-Replay	Basis.	f.	A	100%
2520	Übertragung von Session-Replay-Daten	Basis.	f.	S	100%
Eigenschaften					
3010	Resilienz der Übertragung	Begeist.	f.	S	75%
3020	Batchverarbeitung	Begeist.	f.	S	100%
3100	Anzahl Partnersysteme	Basis.	n. f.	K	66%
3200	Structured Logging	Leistungs.	f.	A+S	100%
Partnersysteme					
5100	Partnersystem <i>Log-Management</i>	Basis.	f.	A+S	100%
5110	Manuelle Analyse <i>Log-Management</i>	Basis.	f.	A+S	100%
5200	Partnersystem <i>Error-Monitoring</i>	Basis.	f.	A+S	100%
5210	Manuelle Analyse <i>Error-Monitoring</i>	Basis.	f.	A+S	100%
5220	Visualisierung <i>Error-Monitoring</i>	Leistungs.	f.	S	100%
5230	Alerting <i>Error-Monitoring</i>	Begeist.	f.	A+S	0%
5300	Partnersystem <i>Tracing</i>	Basis.	f.	A+S	100%
5310	Manuelle Analyse <i>Tracing</i>	Basis.	f.	A+S	100%
5320	Visualisierung <i>Tracing</i>	Basis.	f.	A+S	100%
5400	Partnersystem <i>Metriken</i>	Leistungs.	f.	A+S	100%
5410	Visualisierung <i>Metriken</i>	Leistungs.	f.	A+S	100%
5420	Alerting <i>Metriken</i>	Begeist.	f.	S	0%
5500	Partnersystem <i>Session-Replay</i>	Basis.	f.	A+S	100%
5510	Nachstellung <i>Session-Replay</i>	Basis.	f.	A+S	100%

Tab. 5.1: Tabellarische Bewertung der Anforderungen

Die Anforderung 3010 konnte nicht vollständig erfüllt, da nicht alle Daten, die der Nachvollziehbarkeit dienen, resilient erfasst und übertragen werden. Genauer werden Logs, Metriken und Fehler so resilient behandelt, wie in Unterunterabschnitt 4.4.2.5 beschrieben, jedoch nicht Traces. Grund hierfür ist, dass durch den Einsatz von OpenTelemetry-Komponenten dies nicht oder nicht einfach möglich war. Da somit 3 von 4 Datentypen eine Resilienz aufweisen, wurde die Anforderung mit einem Erfüllungsgrad von 75% bewertet.

Die Anforderung 3100, die Anzahl der Systeme gering zu halten, konnte nicht vollständig umgesetzt werden. Grund hierfür ist, dass zwar auf ein weiteres System zum Error-Monitoring verzichtet werden konnte, aber dennoch 3 zusätzliche Partnersysteme notwendig sind. Somit wurde diese Anforderung als teilweise erfüllt bewertet.

Die Anforderungen 5230 und 5420 wurden nicht erfüllt, da kein Alerting mithilfe von Splunk umgesetzt wurde. Grund hierfür war eine Netzwerkeinschränkung des Kubernetesclusters, die das Übertragen von Daten an außenstehende Systeme nicht einfach möglich machte. Dadurch, dass es sich um zwei Begeisterungsmerkmale handelte, wurde ein Fehlen dieser „Alerting“-Funktionalität akzeptiert.

5.3 Übertragbarkeit

Neben der Erfüllung der gestellten Anforderungen sowie der Aufdeckung von Fehlerszenarien, sollte die erstellten Ergebnisse eine Übertragbarkeit aufweisen. Übertragbarkeit meint hierbei, ob und wie gut sich die ermittelten Ergebnisse auf andere Projekte übertragen lassen.

Zunächst wurde eine Demoanwendung erstellt, anhand dessen die Entwicklung des Konzeptes und der eigentlichen Implementierung erfolgten. Bei dieser Demoanwendung handelt es sich um eine Single-Page-Application, die mit Angular umgesetzt wurde und auf ein Backend zugreift, das mittels einer Microservice-orientierten Architektur umgesetzt wurde. Setzen Aspekte der erstellten Lösung auf diese Randbedingungen, schränken sie die Übertragbarkeit auf andere Systeme ein.

Das Konzept wurde jedoch nicht starr an die Demoanwendung angelehnt, sondern basiert grundlegend auf den zuvor identifizierten fehlenden Informationen bei SPAs und den ermittelten Technologien, um diese Informationslücke zu füllen. Somit weißt das Konzept eine grundlegend gute Übertragbarkeit auf. Anders ist dies jedoch bei der Umsetzung, denn hier wurden einige Aspekte auf Basis der eingesetzten Frameworks umgesetzt. Beispielsweise basiert das Tracing im Backend auf der Unterstützung von OpenTracing im Eclipse-Microprofile-Framework. Des Weiteren basiert die Erfassung von Logs und Fehlern im Frontend auf Angular spezifischen Bibliotheken oder Komponenten.

Eine direkte Bewertung der Übertragbarkeit erfolgt nicht, stattdessen werden die Annahmen erfasst. Die Erfassung der Annahmen erfolgt in Tabelle 5.2. Dabei wird jede Annahme jeweils für das Konzept sowie für die Umsetzung bewertet, dabei werden folgende Bewertungsschlüssel verwendet:

- **ja:** Die Annahme muss erfüllt sein.
- **nein:** Die Annahme muss nicht erfüllt sein.

- **nebensächlich:** Damit ist gemeint, dass die Annahme nicht erfüllt sein muss, jedoch dies bspw. zu einer Minderung der Funktionalität führt. Weiterhin kann hiermit auch gemeint sein, dass die Annahme durch einen äquivalenten Ersatz erfüllt werden kann.

Annahme	Konzept	Umsetzung
Das System kann erweitert werden	ja	ja
Das System kann erweitert werden	ja	ja
Das Frontend ist eine RIA	ja	ja
Das Frontend ist eine SPA	nein	ja
Das Frontend wurde mit Angular erstellt	nein	ja
Das Frontend realisiert einen Wizard	nein	nein
Das Backend folgt einem Microservice-Architektur-Ansatz	nebensächl.	ja
Das Backend wurde mit Java umgesetzt	nein	ja
Das Backend wurde mit Eclipse MicroProfile umgesetzt	nein	nebensächl.
Splunk kann eingesetzt werden	nebensächl.	ja
Jaeger kann eingesetzt werden	nebensächl.	ja
LogRocket kann eingesetzt werden	ja	ja

Tab. 5.2: Annahmen der erstellten Lösungen

5.4 Einfluss für den Nutzer

In diesem Abschnitt ist festzustellen, welchen Einfluss die Lösung für den Nutzer der jeweiligen Webanwendung besitzt. Dabei gilt es bspw. den Aspekt der Ladezeit sowie des erhöhten Datenaufkommens zu beleuchten.

Um die leistungsrelevanten Daten zu erheben, wurden das Frontend jeweils ohne und mit der Lösung mithilfe des „Performance Analysis“-Werkzeugs im Firefox [MM21f] analysiert. Um den Aspekt der Ladezeit, also die Zeit bis die Anwendung für den Nutzer interaktiv wird, zu messen, wurde pauschal die Zeit bis zur ersten ausgehenden Datenabfrage gemessen. Die Ergebnisse sind in Abbildung 5.6 zu betrachten. Dabei wird erkennbar, dass die Lösung mit 1080ms statt 780ms eine leicht erhöhte durchsch. Ladezeit aufweist.

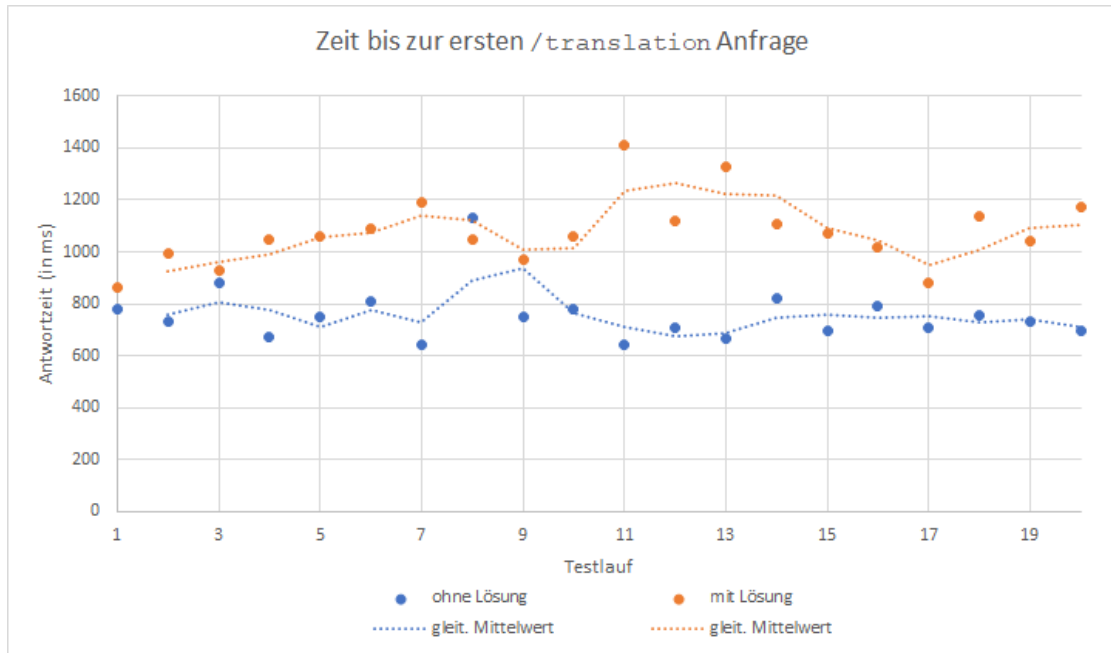


Abb. 5.6: Einfluss für den Nutzer: Zeit bis zur ersten Anfrage

Weiterhin wurde der Aspekt des erhöhten Datenaufkommens näher beleuchtet, indem pro Testlauf jeweils einmal ein Standard-Workflow in der Anwendung durchgespielt worden ist. Am Ende eines Durchlaufs wurde die Anzahl an tatsächlich¹ übertragenden XHR-Bytes notiert, das Resultat kann in Abbildung 5.7 betrachtet werden. Da der Nutzer selbst entscheiden kann, ob LogRocket aktiviert sein soll oder nicht, wurde die Lösung einmal mit aktivierten LogRocket und einmal deaktivierten LogRocket getestet. Weiterhin wurde die Anzahl der Anfragen notiert (vgl. Abbildung 5.8).

Dabei lässt sich feststellen, dass das Datenaufkommen in beiden Fällen kaum erhöht ist. Grund hierfür ist, dass die zusätzlichen Daten durch die automatisch durchgeführte Komprimierung² des Browser, kaum zum Datenaufkommen beitragen. Jedoch ist ein spürbarer Anstieg der Anfragen zu verzeichnen.

Letztendlich lässt sich feststellen, dass der Performance-Einfluss der Lösung marginal und vernachlässigbar ist.

²Durch Komprimierung wie gzip sind die tatsächlich zu übertragenden Bytes geringer als die eigentliche Datenmenge [MM21c].

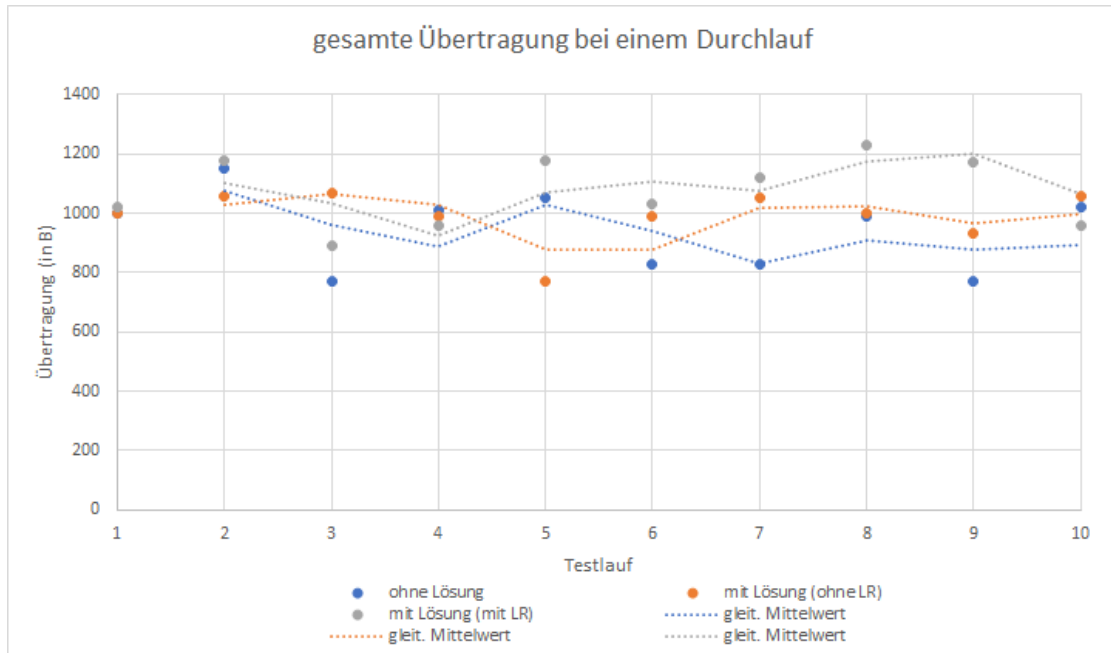


Abb. 5.7: Einfluss für den Nutzer: Gesamte Übertragungsmenge

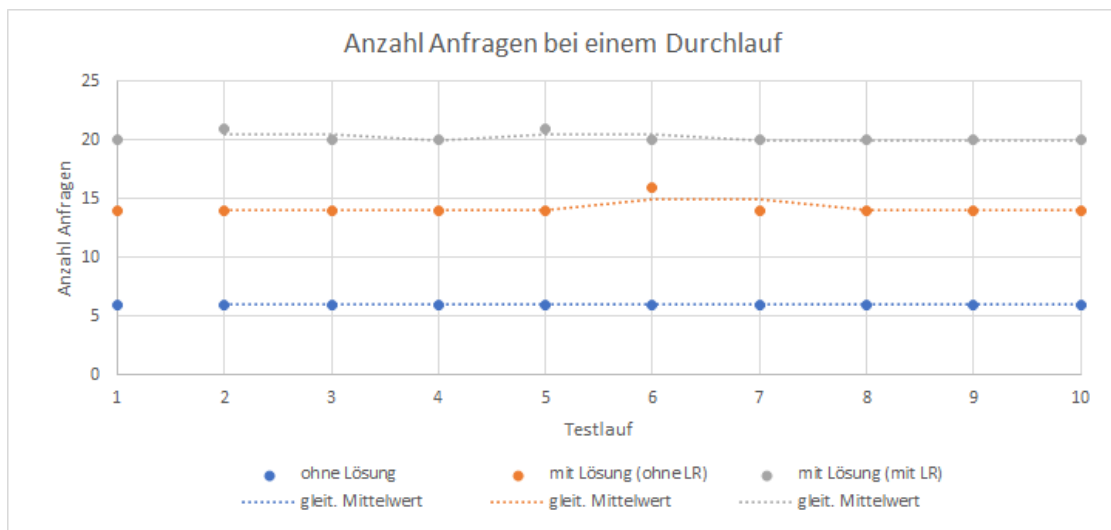


Abb. 5.8: Einfluss für den Nutzer: Anzahl der Anfragen

6 Abschluss

6.1 Zusammenfassung

Ziel der Arbeit war es einen Ansatz zu erstellen, mit dem Betreibern von JavaScript-basierten Webanwendungen eine Nachvollziehbarkeit gewährleistet werden kann. Der Proof-of-Concept, konnte die Mehrheit der gestellten Anforderungen erfüllen sowie zuvor definierte Fehlerszenarien aufgedeckt werden. Weiterhin weist die erstellte Lösung und dabei das Konzept, eine Übertragbarkeit auf andere ähnliche Softwareprojekte auf. Zudem konnte kein signifikanter negativer Einfluss für den Nutzer festgestellt werden. Somit wurde das grundlegende Ziel dieser Arbeit erreicht.

6.2 Fazit

Es konnte ein Kernproblem von Webanwendungen und speziell bei SPAs festgestellt werden, aber es konnten zudem Methoden und Technologien identifiziert werden, die dieses Kernproblem beheben. Weiterhin wurden diese Ansätze erfolgreich eingesetzt und boten einen Mehrwert für Entwickler und Betreiber, der sich nur marginal auf die Performance der Webanwendung niederschlug.

6.3 Ausblick

Wie bei der Recherche zum Stand der Technik zu sehen war, gibt es seit 2020 mit dem OpenTelemetry-Standard eine neue Entwicklung, die das Feld der Nachvollziehbarkeit bzw. der Observability in den nächsten Jahren nachhaltig beeinflussen wird. Sollte der Standard von Herstellern adaptiert werden, könnte dies zu einer höheren Auswahl an Technologien führen, die verwendet werden können, da sie miteinander kompatibel sind. Des Weiteren kann es hierdurch einfacher werden mehrere Observability-Systeme und -Konzepte miteinander zu kombinieren, um eine erhöhte Durchleuchtung zu erreichen. Diese Entwicklung ist in meinen Augen vielversprechend, besonders durch die Unterstützung von führenden Herstellern von Observability-Werkzeugen. Somit sollte dieses Feld in nächster Zeit verfolgt werden.

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 25.02.2021 abgerufen. Dabei wurde für jedes Jahr eine eigene Suche durchgeführt und die Ergebnisanzahl notiert, wobei jede Suche eingeschränkt wurde, dass das Veröffentlichungsdatum im jew. Jahr lag. Für die Suche wurde folgender Suchterm benutzt:

"cross browser" compatibility|incompatibility|inconsistency|XBI

Die jahresbezogene Trefferanzahl (vgl. Tabelle 7.1) soll aufdecken, ob ein Trend in der Literatur zu erkennen ist. Ein schwacher, aber vorhandener Negativtrend konnte festgestellt werden.

Weiterhin lässt sich dieselbe Thematik bei Google Trends [Goo21d] untersuchen. Hierbei wurden die Suchtrends für den Suchterm **cross browser compatability** abgerufen und geplottet (vgl. Abbildung 7.1). Dabei lässt sich ebenso ein Negativtrend erkennen.

Jahr	Suchtreffer
2015	272
2016	228
2017	208
2018	204
2019	172
2020	167

Tab. 7.1: Suchtreffer zu Studien über Browserkompatibilität

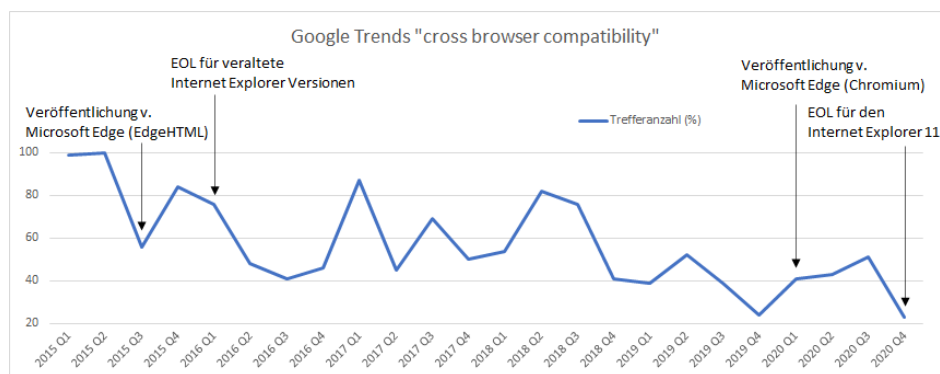


Abb. 7.1: Google Trends zur Browserkompatibilität, angereichert mit [Mic21a]

7.2 Weitere Demonstration

7.2.1 LogRocket

Eine Aufzeichnung des Session-Replays von LogRocket zu einer beispielhaften Sitzung, kann unter [Anhang/session-replay-via-logrocket.mp4](#) betrachtet werden. Das eigentliche Session-Replay erfolgt interaktiv im Browser und Daten wie der DOM, die Konsole und auch Netzwerkaufrufe können vom Entwickler inspiziert werden.

7.2.2 Splunk

Neben Logs wurden in Splunk zudem auch Metriken und Fehler erhoben. Bei den aufgezeichneten Metriken handelt es sich um Beispiele für den in der Demo umgesetzten Anwendungsfall: Die Anzahl an Produkten im Warenkorb und die aufgetretenen Fehler in einer Sitzung. Die Metriken wurden auf einer Seite in Splunk visuell dargestellt, welche in Abbildung 7.2 zu sehen ist.

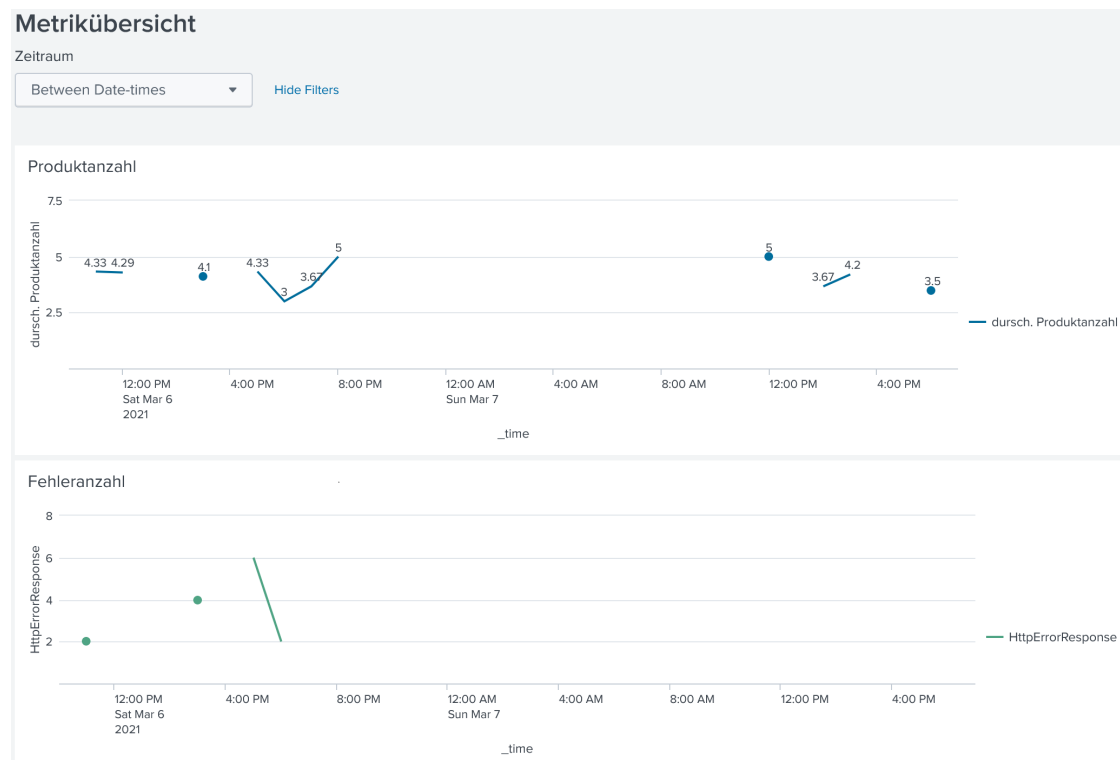


Abb. 7.2: Metrikübersicht in Splunk

Um Fehler zu visualisieren und gruppiert darzustellen, wurde in Splunk eine eigene Seite erstellt, die in Abbildung 7.3 zu betrachten ist. In der Fehlerübersicht lassen sich ein Histogramm, ein Top-Chart sowie die letzten 50 Events im Detail betrachten. Durch diese Seite lassen sich bspw. erhöhte Fehlerraten feststellen und die Häufigkeit von unterschiedlichen Fehlern vergleichen.

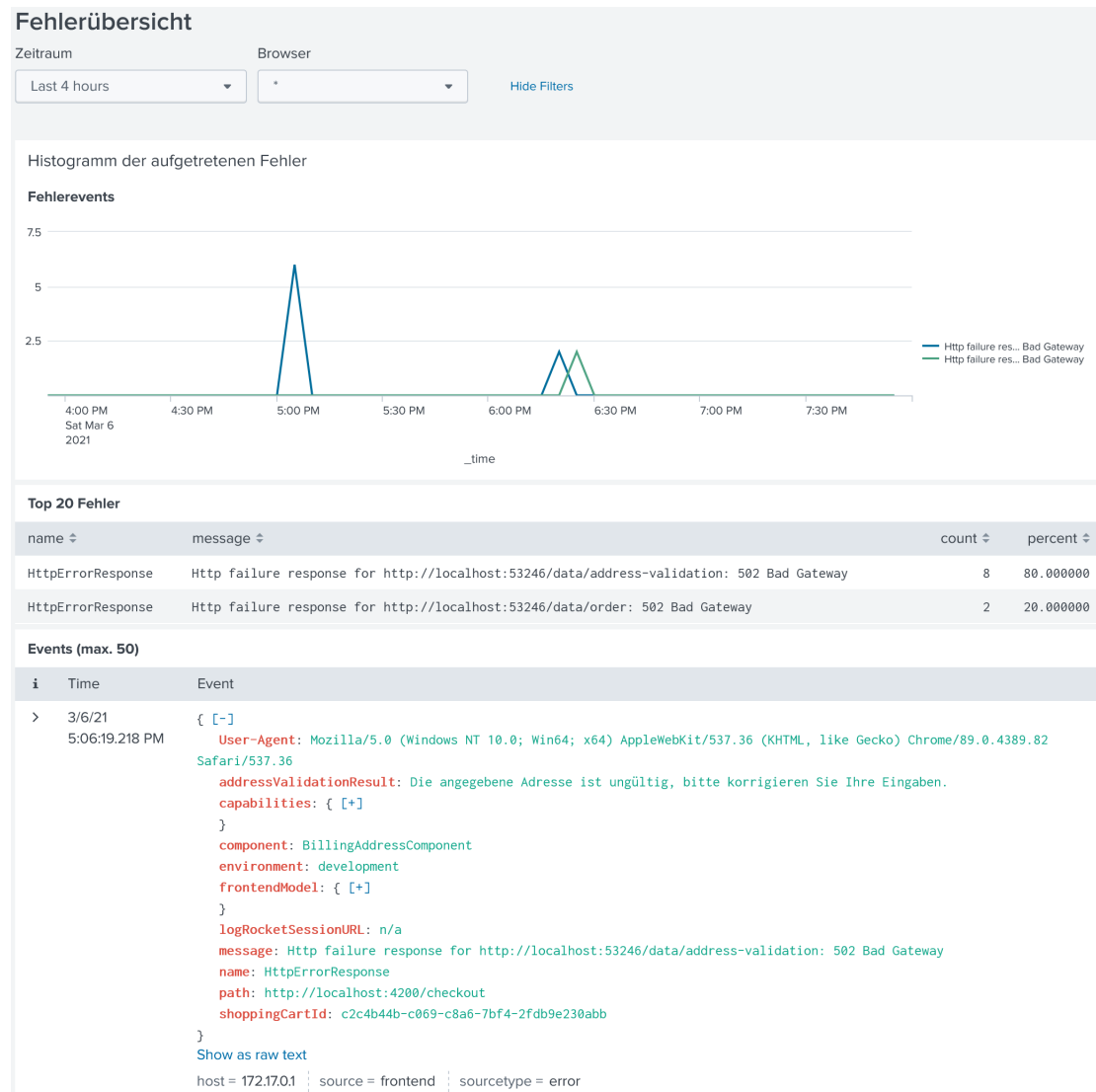


Abb. 7.3: Fehlerübersicht in Splunk

7.2.3 Jaeger

In Jaeger lassen sich neben dem Trace-Gantt-Diagramm auch Traces miteinander vergleichen, sodass Unterschiede in den Aufrufen entdeckt werden können. In den Abbildungen 7.4 und 7.5 ist ein solcher Vergleich zu sehen. Hierbei werden Traces verglichen, die durch eine Anfrage von Warenkorb- und Übersetzungsdaten von Frontend aus resultierten. Dabei divergieren die beiden Traces in der Antwort vom Übersetzungsdienst, denn es wird bei einem Trace dort ein Fehler geworfen.

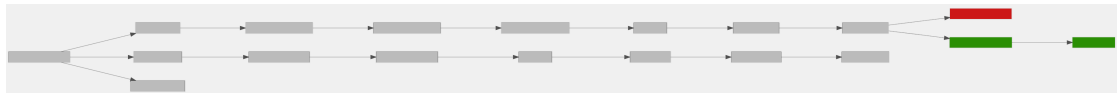


Abb. 7.4: Tracevergleich in Jaeger

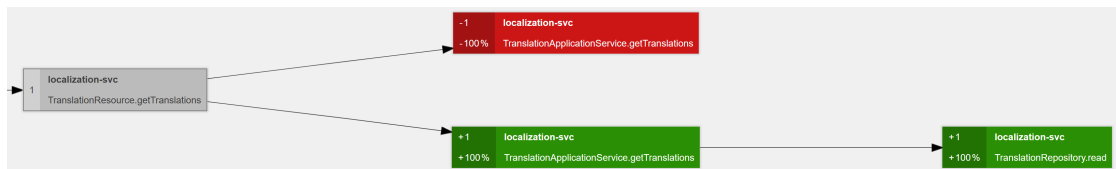


Abb. 7.5: Tracevergleich in Jaeger (Zoom)

Der von Jaeger automatisch erstellte Dienst-Abhängigkeits-Graph kann folgend in Abbildung 7.6 betrachtet werden.

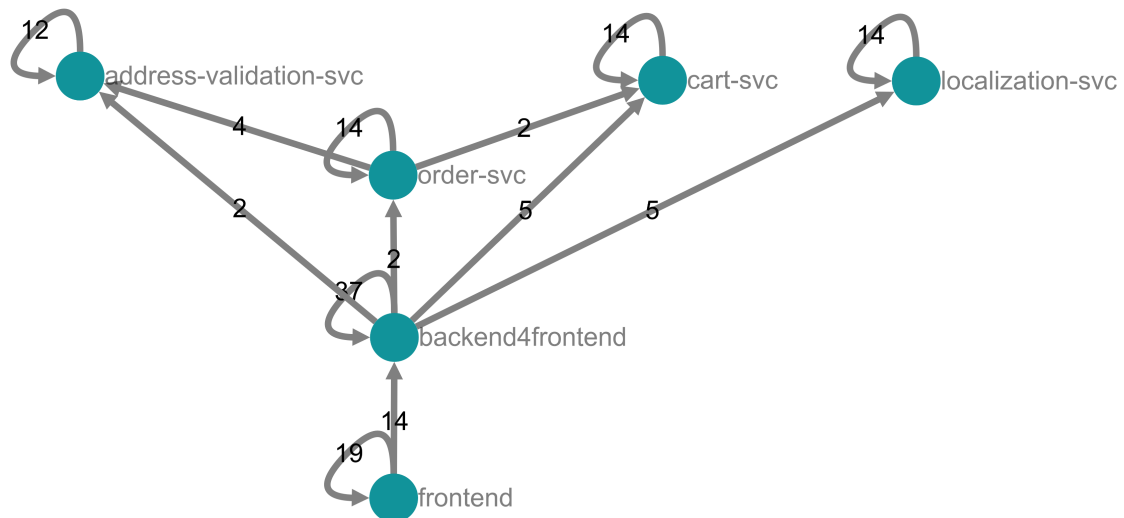


Abb. 7.6: Dienst-Abhängigkeits-Graph in Jaeger

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

Clientseitiges Rendering Der Server stellt dem Client lediglich die Logik und die notwendigen Daten bereit, die eigentliche Inhaltsgenerierung geschieht im Client. Beispiel siehe Unterabschnitt 2.3.1

CNCF Cloud-Native-Computing-Foundation

CORS Cross-Origin Resource Sharing

CSP Content-Security-Policy

gRPC gRPC Remote-Procedure-Call

OTel OpenTelemetry

PoC Proof-of-Concept

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

SPA Single-Page-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. Abschnitt 7.1)	4
2.2	Flowchart über den Ablauf von Ajax-Anfragen mit CORS [Blu15]	7
3.1	Fehlerbericht in der Instagram App [Fac20]	11
3.2	Struktur eines Prometheus-Metrik-Datensatzes [Sri18]	13
3.3	Kausale Beziehung zwischen Spans. Eigene Darstellung.	14
3.4	Zeitliche Beziehung zwischen Spans. Eigene Darstellung.	14
3.5	Komponenten von OpenTelemetry, eigene Darstellung auf Basis von [Ope21c]	16
3.6	Mitschneiden von DOM-Events, Abb. aus [BBKE13]	19
3.7	Abspielen von DOM-Events, Abb. aus [BBKE13]	19
3.8	Abfragebeispiel in Splunk aus [Spl21d]	30
3.9	Dienst-Abhängigkeits-Graph. Quelle: Eigene Darstellung	31
3.10	Trace-Detailansicht. Eigener Screenshot aus Jaeger	31
3.11	Kerninformation eines Issues. Eigener Screenshot aus Sentry	33
3.12	Verlauf der Userinteraktionen. Eigener Screenshot aus Sentry	33
3.13	Ausschnitt eines Session-Replays. Eigener Screenshot aus LogRocket.	34
4.1	Demoanwendung: Deployment-Diagramm. Eigene Darstellung	40
4.2	Demoanwendung: Startseite „Warenkorb“. Eigener Screenshot	41
4.3	Demoanwendung: Seite „Rechnungsadresse“. Eigener Screenshot	42
4.4	Demoanwendung: Seite „Lieferdaten“. Eigener Screenshot	43
4.5	Demoanwendung: Seite „Zahlungsdaten“. Eigener Screenshot	44
4.6	Demoanwendung: Seite „Bestellübersicht“. Eigener Screenshot	45
4.7	Demoanwendung: Finale Seite „Bestellbestätigung“. Eigener Screenshot	46
4.8	Fehlende Texte. Eigener Screenshot	47
4.9	Architektur des Konzeptes. Eigene Darstellung	57
4.10	Ausschnitt des Traces zu Quellcode 4.6	58
4.11	Button zum Öffnen des Dialogs	67
4.12	Einverständnis-Dialog	67
4.13	Architektur des Proof-of-Concepts. Eigene Darstellung	69
5.1	Suche nach Logs zu fehlenden Übersetzungen in Splunk	70
5.2	Suchergebnisse in Jaeger zu spezieller <code>shoppingCartId</code>	71
5.3	Ausschnitt des Traces zur Übersetzungsanfrage in Jaeger	71

5.4	Ausschnitt des Traces zur Bestellanfrage in Jaeger	72
5.5	Ausschnitt des Traces zur Warenkorbanfrage in Jaeger	73
5.6	Einfluss für den Nutzer: Zeit bis zur ersten Anfrage	77
5.7	Einfluss für den Nutzer: Gesamte Übertragungsmenge	78
5.8	Einfluss für den Nutzer: Anzahl der Anfragen	78
7.1	Google Trends zur Browserkompatibilität, angereichert mit [Mic21a] . . .	80
7.2	Metrikübersicht in Splunk	81
7.3	Fehlerübersicht in Splunk	82
7.4	Tracevergleich in Jaeger	83
7.5	Tracevergleich in Jaeger (Zoom)	83
7.6	Dienst-Abhängigkeits-Graph in Jaeger	83

Tabellenverzeichnis

2.1	Übersicht aktueller Browser [Sta21f] [DD20]	5
3.1	Übersicht der untersuchten Technologien	22
3.2	Kategorisierung der untersuchten Technologien	25
3.3	Bewertung der Technologien der Kategorie „Log-Plattformen“	27
3.4	Bewertung der Technologien der Kategorie „Distributed-Tracing-Systeme“	28
3.5	Bewertung der Technologien der Kategorie „Error-Tracking“	29
3.6	Bewertung der Technologien der Kategorie „Session-Replay-Dienste“	29
4.1	Merkmale nach dem Kano-Modell der Kundenzufriedenheit [Kan68]	48
4.2	Kategorien der Anforderungen [SS97]	49
4.3	Quellen der Anforderungen	49
4.4	Beispiel einer Anforderung	50
4.5	Übersicht der ausgewählten Technologien	56
5.1	Tabellarische Bewertung der Anforderungen	74
5.2	Annahmen der erstellten Lösungen	76
7.1	Suchtreffer zu Studien über Browserkompatibilität	80

Quellcodeverzeichnis

4.1	Demoanwendung: Gherkin Definition zum Feature „Warenkorb“	36
4.2	Demoanwendung: Gherkin Definition zum Feature „Rechnungsadresse“	36
4.3	Demoanwendung: Gherkin Definition zum Feature „Lieferadresse“	37
4.4	Demoanwendung: Gherkin Definition zum Feature „Zahlungsdaten“	38
4.5	Demoanwendung: Gherkin Definition zum Feature „Bestellung abschließen“	39
4.6	Beispielhafter Einsatz der @Traced-Annotation	57

4.7	Quellcode des Moduls „app-observability.module.ts“	59
4.8	Datenquelle zum Abrufen und Zusammenführen der Artikeldaten	60
4.9	Service zum Abrufen der Übersetzungsdaten	61
4.10	Ausschnitt des Hauptmoduls <code>app.module.ts</code>	62
4.11	Implementierung des <code>NGXLoggerMonitor</code> -Interfaces	63
4.12	ErrorHandler zum Abfangen und Weiterleiten von aufgetretenen Fehlern .	64
4.13	Implementierung des <code>SplunkForwardingService</code>	65
4.14	Beispiel eines Batches	66
4.15	Angular HTML-Template der Einverständnis-Komponente	67
4.16	Initialisierung von LogRocket in der Einverständnis-Komponente	68

Literaturverzeichnis

- [ABC⁺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
- [Ado21] ADOBE: *Adobe Analytics for deeper insights*. <https://www.adobe.com/analytics/adobe-analytics.html>, 2021. – [Online; abgerufen am 17.02.2021]
- [All02] ALLAIRE, Jeremy: Macromedia Flash MX-A next-generation rich client. In: *Macromedia white paper* (2002), S. 1–2
- [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
- [Apa21] APACHE SOFTWARE FOUNDATION: *Apache Thrift - Home*. <https://thrift.apache.org/>, 2021. – [Online; abgerufen am 02.03.2021]
- [App21a] APPDYNAMICS: *Application Performance Management (APM) | Produkt | AppDynamics*. https://www.appdynamics.com/de_de/product/application-performance-management, 2021. – [Online; abgerufen am 17.02.2021]
- [App21b] APPLE: *WebKit*. <https://webkit.org/>, 2021. – [Online; abgerufen am 02.03.2021]
- [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⁺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, 2015. – [Online; abgerufen am 14.03.2021]
- [Bra19] BRANDHORST, Johan: *The state of gRPC in the browser / gRPC*. <https://github.com/grpc/grpc.io/blob/main/content/en/blog/state-of-grpc-web.md>, 2019. – [Online; abgerufen am 02.03.2021]
- [Bro21] BROADCAST: *Application Performance Management*. <https://www.broadcom.com/products/software/aiops/application-performance-management>, 2021. – [Online; abgerufen am 17.02.2021]
- [BT19] BRUHIN, Florian ; TAVERNINI, Luca: *Crashbin – Dokumentation*. Hochschule für Technik Rapperswil & Fachhochschule Ostschweiz, 2019
- [CCP12] CINQUE, Marcello ; COTRONEO, Domenico ; PECCHIA, Antonio: Event logs for the analysis of software failures: A rule-based approach. In: *IEEE Transactions on Software Engineering* 39 (2012), Nr. 6, S. 806
- [CDM⁺12] CHOUDHARY, Suryakant ; DINCTURK, Mustafa E. ; MIRTAHERI, Seyed M. ; MOOSAVI, Ali ; BOCHMANN, Gregor von ; JOURDAN, Guy-Vincent ; ONUT, Iosif V.: Crawling rich internet applications: the state of the art. In: *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, 2012, S. 146–160
- [ČF17] ČEGAN, Lukáš ; FILIP, Petr: Advanced web analytics tool for mouse tracking and real-time data processing. In: *2017 IEEE 14th International Scientific Conference on Informatics* IEEE, 2017, S. 431–435
- [CGL⁺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
- [CGM14] CASTELEYN, Sven ; GARRIGÓS, Irene ; MAZÓN, Jose-Norberto: Ten years of rich internet applications: A systematic mapping study, and beyond. In: *ACM Transactions on the Web (TWEB)* 8 (2014), Nr. 3, S. 1–3
- [CPO14] CHOUDHARY, Shauvik R. ; PRASAD, Mukul R. ; ORSO, Alessandro: XPERT: 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

- [CSP12] *Kapitel 20.* In: CHUVAKIN, Anton ; SCHMIDT, Kevin ; PHILLIPS, Chris: *Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management.* Newnes, 2012, S. 369–380
- [Dat21] DATADOG: *Cloud Monitoring as a Service / Datadog.* <https://www.datadoghq.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [DCZ11] DONG, Shuxia ; CHENG, Chen ; ZHOU, Yi: Research on AJAX technology application in web development. In: *2011 International Conference on E-Business and E-Government (ICEE)* IEEE, 2011, S. 1–3
- [DD18] DISSANAYAKE, Nalaka R. ; DIAS, K.: Rich web-based applications: an umbrella term with a definition and taxonomies for development techniques and technologies. In: *Int. J. Future Comput. Commun* 7 (2018), Nr. 1, S. 14–20
- [DD20] DOWDEN, Martine ; DOWDEN, Michael: Compatibility and Defaults. In: *Architecting CSS.* Springer, 2020, S. 127–128
- [Doc20] DOCKER: *Docker overview / Docker Documentation.* <https://docs.docker.com/get-started/overview/>, 2020. – [Online; abgerufen am 15.02.2021]
- [Dog20] *Kapitel 1.* In: DOGUHAN, Uluca: *Angular for Enterprise-Ready Web Applications : Build and Deliver Production-grade and Cloud-scale Evergreen Web Apps with Angular 9 and Beyond, 2nd Edition..* Bd. Second edition. Packt Publishing, 2020. – ISBN 9781838648800, S. 4–9
- [Dyn20] DYNATRACE: *Application performance monitoring (APM) / Dynatrace.* <https://www.dynatrace.com/platform/application-performance-monitoring/>, 2020. – [Online; abgerufen am 21.02.2021]
- [Dyn21] DYNATRACE: *The Leader in Cloud Monitoring / Dynatrace.* <https://www.dynatrace.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [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
- [Ecl21] ECLIPSE FOUNDATION: *MicroProfile / projects.eclipse.org.* <https://projects.eclipse.org/projects/technology.microprofile>, 2021. – [Online; abgerufen am 10.02.2021]
- [Fac20] FACEBOOK: *Instagram App Screenshot.* <https://www.instagram.com/>, 2020

- [Fac21] FACEBOOK: *React - A JavaScript library for building user interfaces*. <https://reactjs.org>, 2021. – [Online; abgerufen am 07.03.2021]
- [FČ18] FILIP, Petr ; ČEGAN, Lukáš: Webalyt: Implemetation of architecture for capturing web user behaviours with feedback propagation. In: *2018 28th International Conference Radioelektronika IEEE*, 2018, S. 1–5
- [FEH⁺14] FATEMA, Kaniz ; EMEAKAROHA, Vincent C. ; HEALY, Philip D. ; MORRISON, John P. ; LYNN, Theo: A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. In: *Journal of Parallel and Distributed Computing* 74 (2014), Nr. 10, S. 2918–2933
- [Fil20] FILIPE, Ricardo Ângelo S.: *Client-Side Monitoring of Distributed Systems*, Universidade de Coimbra, Diss., 2020
- [Flu21a] FLUENTD PROJECT: *Fluentd / Open Source Data Collector / Unified Logging Layer*. <https://www.fluentd.org/>, 2021. – [Online; abgerufen am 17.02.2021]
- [Flu21b] FLUENTD PROJECT: *Why use Fluentd? / Fluentd*. <https://www.fluentd.org/why>, 2021. – [Online; abgerufen am 19.02.2021]
- [FN07] FARRELL, Jason ; NEZLEK, George S.: Rich internet applications the next stage of application development. In: *2007 29th International Conference on Information Technology Interfaces IEEE*, 2007, S. 413–418
- [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
- [FRSF10] FRATERNALI, Piero ; ROSSI, Gustavo ; SÁNCHEZ-FIGUEROA, Fernando: Rich Internet Applications. In: *IEEE Internet Computing* 14 (2010), Nr. 3, S. 9–12
- [Ful21] FULLSTORY: *FullStory: Build a More Perfect Digital Experience / FullStory*. <https://www.fullstory.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [Fun21a] FUNCTIONAL SOFTWARE: *About Sentry / Sentry*. <https://sentry.io/about/>, 2021. – [Online; abgerufen am 17.02.2021]
- [Fun21b] FUNCTIONAL SOFTWARE: *getsentry/sentry-javascript: Official Sentry SDKs for Javascript*. <https://github.com/getsentry/sentry-javascript>, 2021. – [Online; abgerufen am 07.03.2021]

- [Fun21c] FUNCTIONAL SOFTWARE: *Self-Hosted Sentry / Sentry Developer Documentation*. <https://develop.sentry.dev/self-hosted/>, 2021. – [Online; abgerufen am 07.03.2021]
- [Gar20] GARTNER: *Gartner Magic Quadrant for Application Performance Monitoring*. <https://www.gartner.com/en/documents/3983892/magic-quadrant-for-application-performance-monitoring>, 2020. – [Online; abgerufen am 21.02.2021]
- [Gar21] GARTNER: *Web and Mobile App Analytics Reviews and Ratings / Gartner Peer Insights*. <https://www.gartner.com/reviews/market/web-mobile-app-analytics>, 2021. – [Online; abgerufen am 17.02.2021]
- [GB21] GREIF, Sacha ; BENITTE, Raphaël: *State of JS 2020: Front-end Frameworks*. <https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>, 2021. – [Online; abgerufen am 07.03.2021]
- [Git21] GITHUB: *About / GitHub*. <https://github.com/about>, 2021. – [Online; abgerufen am 19.02.2021]
- [Goo21a] GOOGLE: *Analytics Tools & Solutions for Your Business - Google Analytics*. <https://marketingplatform.google.com/about/analytics/>, 2021. – [Online; abgerufen am 17.02.2021]
- [Goo21b] GOOGLE: *Angular*. <https://angular.io>, 2021. – [Online; abgerufen am 07.03.2021]
- [Goo21c] GOOGLE: *Angular - The RxJS library*. <https://angular.io/guide/rx-library>, 2021. – [Online; abgerufen am 05.03.2021]
- [Goo21d] GOOGLE: *cross browser compatibility - Explore - Google Trends*. <https://trends.google.com/trends/explore?date=2015-01-01%202020-12-31&q=cross%20browser%20compatibility>, 2021. – [Online; abgerufen am 25.02.2021]
- [HHMO17] HEGER, Christoph ; HOORN, André van ; MANN, Mario ; OKANOVIĆ, Dušan: Application performance management: State of the art and challenges for the future. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, 2017, S. 429–432
- [HMQLJ21] HERNÁNDEZ, Cristian M. ; MARTÍNEZ, Alexandra ; QUESADA-LÓPEZ, Christian ; JENKINS, Marcelo: Comparison of End-to-End Testing Tools for Microservices: A Case Study. In: *International Conference on Information Technology & Systems* Springer, 2021, S. 407–416
- [Hög20] HÖGLUND, Jonas: An Analysis Of A Distributed Tracing Systems Effect On Performance. (2020)

- [HZH⁺17] HE, Pinjia ; ZHU, Jieming ; HE, Shilin ; LI, Jian ; LYU, Michael R.: Towards automated log parsing for large-scale log data analysis. In: *IEEE Transactions on Dependable and Secure Computing* 15 (2017), Nr. 6, S. 931–944
- [IF14] Kapitel 1s. In: IDO FLATOW, Gil F.: *Introducing Single Page Applications*. Berkeley, CA : Apress, 2014. – ISBN 978–1–4302–6674–7, S. 3–13
- [Ins21] INSPECTLET: *Inspectlet - Session Recording, Website Heatmaps, Javascript A/B Testing, Error Logging, Form Analytics*. <https://www.inspectlet.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [Ivy13] IVY WIGMORE: *What is Gartner? - Definition from WhatIs.com*. <https://whatis.techtarget.com/definition/Gartner>, 2013. – [Online; abgerufen am 17.02.2021]
- [Jae21a] JAEGER AUTHORS: *Client Libraries - Jaeger documentation*. <https://www.jaegertracing.io/docs/1.21/client-libraries/>, 2021. – [Online; abgerufen am 15.02.2021]
- [Jae21b] JAEGER AUTOREN: *Jaeger: open source, end-to-end distributed tracing*. <https://www.jaegertracing.io/>, 2021. – [Online; abgerufen am 14.03.2021]
- [Jae21c] JAEGER AUTOREN: *OpenTelemetry - Jaeger documentation*. <https://www.jaegertracing.io/docs/1.21/opentelemetry/>, 2021. – [Online; abgerufen am 19.02.2021]
- [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
- [Kau07] Kapitel 1. In: KAUSHIK, Avinash: *Web analytics: an hour a day*. John Wiley & Sons, 2007, S. 1–10
- [Kha17] KHAN, Asif: Key characteristics of a container orchestration platform to enable a modern application. In: *IEEE cloud Computing* 4 (2017), Nr. 5, S. 42–48

- [KKV19] KALUŽA, Marin ; KALANJ, Marijana ; VUKELIĆ, Bernard: A comparison of back-end frameworks for web application development. In: *Zbornik veleučilišta u rijeci* 7 (2019), Nr. 1, S. 317–332
- [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
- [Lin20] LINUX FOUNDATION: *What is Kubernetes? / Kubernetes*. <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>, 2020. – [Online; abgerufen am 15.02.2021]
- [Lin21] LINUX FOUNDATION: *About gRPC / gRPC*. <https://grpc.io/about/>, 2021. – [Online; abgerufen am 15.02.2021]
- [LLG⁺19] LI, Wubin ; LEMIEUX, Yves ; GAO, Jing ; ZHAO, Zhuofeng ; HAN, Yanbo: Service mesh: Challenges, state of the art, and future research opportunities. In: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)* IEEE, 2019, S. 124–127
- [Lof17] LOFFAY, Pavol: *Usage in a browser - Issue #109 - jaegertracing/jaeger-client-node*. <https://github.com/jaegertracing/jaeger-client-node/issues/109>, 2017. – [Online; abgerufen am 02.03.2021]
- [Log21a] LOGROCKET: *LogRocket / Logging and Session Replay for JavaScript Apps*. <https://logrocket.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [Log21b] LOGROCKET: *Performance*. <https://docs.logrocket.com/docs/performance>, 2021. – [Online; abgerufen am 07.03.2021]
- [LVG⁺18] LEZNIK, Mark ; VOLPERT, Simon ; GRIESINGER, Frank ; SEYBOLD, Daniel ; DOMASCHKA, Jörg: Done yet? A critical introspective of the cloud management toolbox. In: *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)* IEEE, 2018, S. 1–8
- [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
- [Men19] MENESES, Luis: Netlytic. In: *Early Modern Digital Review* 2 (2019), Nr. 1
- [MHDJ09] MAES, Wim ; HEYMAN, Thomas ; DESMET, Lieven ; JOOSEN, Wouter: Browser protection against cross-site request forgery. In: *Proceedings of the first ACM workshop on Secure execution of untrusted code*, 2009, S. 3–10

- [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/microsoft-365-apps-say-farewell-to-internet-explorer-11-and/ba-p/1591666>, 2020. – Pressemitteilung, [Online; abgerufen am 25.02.2021]
- [Mic20b] MICROSOFT: *New Microsoft Edge to replace Microsoft Edge Legacy with April's Windows 10 Update Tuesday release*. <https://techcommunity.microsoft.com/t5/microsoft-365/new-microsoft-edge-to-replace-microsoft-edge-legacy-with-april-s/ba-p/2114224>, 2020. – Pressemitteilung, [Online; abgerufen am 25.02.2021]
- [Mic21a] MICROSOFT: *Lifecycle FAQ - Internet Explorer and Edge | Microsoft Docs*. <https://docs.microsoft.com/en-us/lifecycle/faq/internet-explorer-microsoft-edge>, 2021. – [Online; abgerufen am 25.02.2021]
- [Mic21b] MICROSOFT: *What is Azure Application Insights? - Azure Monitor | Microsoft Docs*. <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview>, 2021. – [Online; abgerufen am 17.02.2021]
- [MM21a] MOZILLA ; MITWIRKENDE individuelle: *Ajax - Developer guides | MDN*. <https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX>, 2021. – [Online; abgerufen am 11.02.2021]
- [MM21b] MOZILLA ; MITWIRKENDE individuelle: *console - Web APIs | MDN*. <https://developer.mozilla.org/en-US/docs/Web/API/Console>, 2021. – [Online; abgerufen am 14.03.2021]
- [MM21c] MOZILLA ; MITWIRKENDE individuelle: *Content-Encoding - HTTP | MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding>, 2021. – [Online; abgerufen am 07.03.2021]
- [MM21d] MOZILLA ; MITWIRKENDE individuelle: *Content Security Policy (CSP) - HTTP | MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP/>, 2021. – [Online; abgerufen am 14.03.2021]
- [MM21e] MOZILLA ; MITWIRKENDE individuelle: *Cross-Origin Resource Sharing (CORS) - HTTP | MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>, 2021. – [Online; abgerufen am 14.03.2021]
- [MM21f] MOZILLA ; MITWIRKENDE individuelle: *Performance Analysis - Firefox Developer Tools | MDN*. https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor/Performance_Analysis, 2021. – [Online; abgerufen am 07.03.2021]

- [New20] NEW RELIC: *New Relic APM / New Relic*. <https://newrelic.com/products/application-monitoring/features>, 2020. – [Online; abgerufen am 21.02.2021]
- [New21] NEW RELIC: *New Relic / Deliver more perfect software*. <https://newrelic.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [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⁺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 CONSORTIUM: *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 07.03.2021]
- [Ope20a] OPENTELEMETRY: *OpenTelemetry Community Members*. <https://github.com/open-telemetry/community/blob/01acd5d2c39b764554ac3d87a64a18a09250e9a3/community-members.md>, 2020. – [Online; abgerufen am 22.02.2021]
- [Ope20b] OPENTRACING: *What is Distributed Tracing?* <https://opentracing.io/docs/overview/what-is-tracing/>, 2020. – [Online; abgerufen am 07.03.2021]
- [Ope21a] OPENCENSUS: *OpenCensus*. <https://opencensus.io/introduction/#overview>, 2021. – [Online; abgerufen am 07.03.2021]

- [Ope21b] OPENTELEMETRY: *open-telemetry/opentelemetry-js: OpenTelemetry JavaScript API and SDK*. <https://github.com/open-telemetry/opentelemetry-js>, 2021. – [Online; abgerufen am 15.02.2021]
- [Ope21c] OPENTELEMETRY: *OpenTelemetry Specification / Overview*. <https://github.com/open-telemetry/opentelemetry-specification/blob/168aa71131420f58d3428e206b84a69b97dd7a07/specification/overview.md>, 2021. – [Online; abgerufen am 07.03.2021]
- [Ope21d] OPENTELEMETRY: *Tracking OTel Spec Issues for GA*. <https://github.com/open-telemetry/opentelemetry-specification/issues/1118>, 2021. – [Online; abgerufen am 01.03.2021]
- [Ope21e] 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]
- [Ope21f] OPENTRACING: *The OpenTracing Semantic Specification*. <https://github.com/opentracing/specification/blob/c064a86b69b9d170ace3f4be7dbacf47953f9604/specification.md>, 2021. – [Online; abgerufen am 07.03.2021]
- [Ora21] ORACLE: *Java Debug Wire Protocol*. <https://download.java.net/java/GA/jdk14/docs/specs/jdwp/jdwp-spec.html>, 2021. – [Online; abgerufen am 07.03.2021]
- [OTMC11] OYAMA, Katsunori ; TAKEUCHI, Atsushi ; MING, Hua ; CHANG, Carl K.: A Concept Lattice for Recognition of User Problems in Real User Monitoring. In: *2011 18th Asia-Pacific Software Engineering Conference IEEE*, 2011, S. 163–170
- [PCQ⁺18] PICORETI, Rodolfo ; CARMO, Alexandre P. ; QUEIROZ, Felipe M. ; GARCIA, Anilton S. ; VASSALLO, Raquel F. ; SIMEONIDOU, Dimitra: Multilevel observability in cloud orchestration. In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech) IEEE*, 2018, S. 776–784
- [PLSC05] PRECIADO, Juan C. ; LINAJE, Marino ; SANCHEZ, Fernando ; COMAI, Sara: Necessity of methodologies to model Rich Internet Applications. In: *Seventh IEEE International Symposium on Web Site Evolution IEEE*, 2005, S. 7–13
- [Pow06] POWERS, Shelley: *Learning JavaScript*. O'Reilly Media Inc., 2006 (Java Series). – ISBN 9780596527464

- [Pro21] PROMETHEUS AUTOREN: *Prometheus - Monitoring system & time series database*. <https://prometheus.io/>, 2021. – [Online; abgerufen am 17.02.2021]
- [PSF04] PHIPPEN, Andrew ; SHEPPARD, L. ; FURNELL, Steven: A practical evaluation of Web analytics. In: *Internet Research* (2004)
- [PSM⁺20] *Kapitel 3*. In: PARKER, Austin ; SPOONHOWER, Daniel ; MACE, Jonathan ; SIGELMAN, Ben ; ISAACS, Rebecca: *Distributed tracing in practice: Instrumenting, analyzing, and debugging microservices*. 1. O'Reilly Media, 2020. – ISBN 978-1-492-05663-8, S. 34-43
- [RAF19] ROCHIM, Adian F. ; AZIZ, Mukhlis A. ; FAUZI, Adnan: Design log management system of computer network devices infrastructures based on ELK stack. In: *2019 International Conference on Electrical Engineering and Computer Science (ICECOS)* IEEE, 2019, S. 338-342
- [Ran09] RANGANATHAN, Arun: *cross-site xmlhttprequest with CORS*. <https://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/>, 2009. – [Online; abgerufen am 07.03.2021]
- [Rel21] RELIC, New: *Why the Future Is Open, Connected, and Programmable*. <https://newrelic.com/resources/ebooks/what-is-observability>, 2021
- [RIS18] REYNOLDS, Christopher R. ; ISLAM, Suhail A. ; STERNBERG, Michael J.: EzMol: a web server wizard for the rapid visualization and image production of protein and nucleic acid structures. In: *Journal of molecular biology* 430 (2018), Nr. 15, S. 2244-2248
- [Rol21] ROLLBAR: *Rollbar - Error Tracking Software for JavaScript, PHP, Ruby, Python and more*. <https://rollbar.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [RSJ⁺12] RABL, Tilmann ; SADOOGHI, Mohammad ; JACOBSEN, Hans-Arno ; GÓMEZ-VILLAMOR, Sergio ; MUNTÉS-MULERO, Victor ; MANKOWSKII, Serge: Solving big data challenges for enterprise application performance management. In: *Proceedings of the VLDB Endowment* 5 (2012), Nr. 12
- [SBB⁺10] SIGELMAN, Benjamin H. ; BARROSO, Luiz A. ; BURROWS, Mike ; STEPHENSON, Pat ; PLAKAL, Manoj ; BEAVER, Donald ; JASPAN, Saul ; SHANBHAG, Chandan: *Dapper, a large-scale distributed systems tracing infrastructure* / Google. 2010. – Forschungsbericht
- [SC05] SERRANO, Nicolas ; CIORDIA, Ismael: Bugzilla, ITracker, and other bug trackers. In: *IEEE software* 22 (2005), Nr. 2, S. 11-13

- [SCF15] SOUSA, Tiago B. ; CORREIA, Filipe F. ; FERREIRA, Hugo S.: Patterns for software orchestration on the cloud. In: *Proceedings of the 22nd Conference on Pattern Languages of Programs*, 2015, S. 1–12
- [SF16] SCHUSTER, Christopher ; FLANAGAN, Cormac: Reactive programming with reactive variables. In: *Companion Proceedings of the 15th International Conference on Modularity*, 2016, S. 29–33
- [SM21] SHARMA, Rahul ; MATHUR, Akshay: Logs, Request Tracing, and Metrics. In: *Traefik API Gateway for Microservices*. Springer, 2021, S. 127–129
- [Sma21] SMARTBEAR SOFTWARE: *Gherkin Syntax*. <https://cucumber.io/docs/gherkin/>, 2021. – [Online; abgerufen am 07.03.2021]
- [Spl21a] SPLUNK: *HTTP Event Collector REST API endpoints - Splunk Documentation*. <https://docs.splunk.com/Documentation/Splunk/8.1.2/Data/HECRESTendpoints>, 2021. – [Online; abgerufen am 21.02.2021]
- [Spl21b] SPLUNK: *Splunk APM | DevOps | Splunk*. https://www.splunk.com/en_us/software/splunk-apm.html, 2021. – [Online; abgerufen am 17.02.2021]
- [Spl21c] SPLUNK: *Splunk SPL for SQL users - Splunk Documentation*. <https://docs.splunk.com/Documentation/Splunk/8.1.2/SearchReference/SQLtoSplunk>, 2021. – [Online; abgerufen am 21.02.2021]
- [Spl21d] SPLUNK: *Use the search language - Splunk Documentation*. <https://docs.splunk.com/Documentation/Splunk/8.1.2/SearchTutorial/Usethesearchlanguage>, 2021. – [Online; abgerufen am 21.02.2021]
- [Sri18] Kapitel 4. In: SRIDHARAN, Cindy: *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media, 2018, S. 17–26
- [SS97] Kapitel Introduction. In: SOMMERVILLE, Ian ; SAWYER, Pete: *Requirements engineering: a good practice guide*. John Wiley and Sons, 1997, S. 7–8
- [Sta21a] STACKSHARE: *Jaeger vs Zipkin | What are the differences?* <https://stackshare.io/stackups/jaeger-vs-zipkin>, 2021. – [Online; abgerufen am 19.02.2021]
- [Sta21b] STACKSHARE: *What are the best Exception Monitoring Tools?* <https://stackshare.io/exception-monitoring>, 2021. – [Online; abgerufen am 17.02.2021]
- [Sta21c] STACKSHARE: *What are the best Monitoring Tools?* <https://stackshare.io/monitoring>, 2021. – [Online; abgerufen am 17.02.2021]

- [Sta21d] STACKSHARE: *What are the best Performance Monitoring Tools?* <https://stackshare.io/performance-monitoring>, 2021. – [Online; abgerufen am 17.02.2021]
- [Sta21e] STACKSHARE: *What are the best User-Feedback-as-a-Service Tools?* <https://stackshare.io/user-feedback-as-a-service>, 2021. – [Online; abgerufen am 17.02.2021]
- [Sta21f] 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
- [TM11] TAIVALSAARI, Antero ; MIKKONEN, Tommi: The web as an application platform: The saga continues. In: *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications* IEEE, 2011, S. 170–174
- [Tra21] TRACKJS: *JavaScript Error Logging - TrackJS*. <https://trackjs.com/>, 2021. – [Online; abgerufen am 17.02.2021]
- [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
- [Vet20] VETHANAYAGAM, Suhanthan: Threat Identification from Access Logs Using Elastic Stack. (2020), November
- [W3C06] W3C: WORLD WIDE WEB CONSORTIUM: *The XMLHttpRequest Object*. <https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>, 2006. – [Online; abgerufen am 07.03.2021]
- [W3C20] W3C: WORLD WIDE WEB CONSORTIUM: *Accessible Rich Internet Applications (WAI-ARIA) 1.1*. <https://www.w3.org/TR/wai-aria/>, 2020. – [Online; abgerufen am 25.02.2021]
- [W3C21] W3C: WORLD WIDE WEB CONSORTIUM: *About W3C Standards*. <https://www.w3.org/standards/about.html>, 2021. – [Online; abgerufen am 07.03.2021]

- [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
- [YM21] YOU, Evan ; MITWIRKENDE individuelle: *Vue.js*. <https://vuejs.org/>, 2021. – [Online; abgerufen am 07.03.2021]
- [YZS16] YU, Fang ; ZHUANG, Weijin ; SUN, Mingyang: Research and application of operating monitoring and evaluation for dispatching automation and control system. In: *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* IEEE, 2016, S. 1638–1641
- [ZHF⁺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: *OpenZipkin · A distributed tracing system*. <https://zipkin.io/>, 2021. – [Online; abgerufen am 14.03.2021]