

Thesis

Nachvollziehbarkeit von Nutzerinteraktion und Anwendungsverhalten am Beispiel JavaScript-basierter Webapplikationen

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

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

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

Dortmund, 15. Januar 2021

Kurzfassung

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

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

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

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

Abstract

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

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

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

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

Inhaltsverzeichnis

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
2.1.2	JavaScript	5
2.2	Clientbasierte Webapplikationen	6
2.2.1	JavaScript-basierte Webapplikationen	6
2.2.2	Single-Page-Applications	6
2.3	Nachvollziehbarkeit	7
2.3.1	Nutzen	7
2.3.2	Nachvollziehbarkeit bei SPAs	7
2.3.3	Browserbedingte Hürden	8
2.3.3.1	Cross-Origin Resource Sharing (CORS)	8
2.3.3.2	Content-Security-Policy	9
2.3.4	Logdaten	9
2.3.5	Fernzugriff	9
3	Methoden und Praktiken	10
3.1	Methoden	10
3.1.1	Logging	10
3.1.1.1	Structured Logging	10
3.1.2	Metriken	10
3.1.3	Tracing	11
3.1.4	Fehlerberichte	12
3.2	Praktiken aus der Wirtschaft	12
3.2.1	Observability	12

3.2.2	System Monitoring	13
3.2.3	Log Management	13
3.2.4	Application Performance Monitoring (APM)	13
3.2.5	Real User Monitoring (RUM)	14
3.2.6	Synthetic Monitoring	14
3.2.7	Error/Crash Monitoring	14
3.2.8	Session-Replay	14
3.3	Werkzeuge und Technologien	15
3.3.1	Fachpraxis	15
3.3.1.1	OpenTelemetry	15
3.3.1.2	New Relic	16
3.3.1.3	Dynatrace	16
3.3.1.4	Sentry	17
3.3.1.5	LogRocket	18
3.3.1.6	Splunk	18
3.3.1.7	Honeycomb	19
3.3.1.8	Jaeger	20
3.3.1.9	Weiteres	20
3.3.2	Literatur	21
3.3.2.1	TraVista	21
3.3.2.2	„Bedeutung von Telemetry für den SDLC“	22
3.3.2.3	Timelapse	23
3.3.2.4	FAME	24
3.3.2.5	The Kaiju Project	25
3.3.2.6	Fazit zur Literatur	26
4	Erstellung Proof-of-Concept	27
4.1	Anforderungen	27
4.1.1	Definitionen	27
4.1.2	Anforderungsanalyse	28
4.1.3	Anforderungsliste	29
4.1.3.1	Grundanforderungen	29
4.1.3.2	Funktionsumfang	30
4.1.3.3	Eigenschaften	32
4.1.3.4	Partnersysteme	32
4.2	Vorstellung der Demoanwendung	34
4.2.1	Verhaltensdefinition	34
4.2.2	Backend	38
4.2.3	Frontend	39
4.2.4	Fehlerszenarien	43
4.2.4.1	„Keine Übersetzungen“	43
4.2.4.2	„Gültige Straßen sind ungültig“	43

Inhaltsverzeichnis

4.2.4.3	„Gültige Hausnummern sind ungültig“	43
4.2.4.4	„Gültige Städte sind ungültig“	44
4.2.4.5	„Ungültige Adressen sind gültig“	44
4.2.4.6	„Vor- und Nachnamen werden abgeschnitten“	44
4.2.4.7	„Falsche Zahlungsart“	44
4.2.4.8	„Lange Verarbeitung“	44
4.2.5	Repräsentation	45
4.3	Konzept	46
4.3.1	Datenverarbeitung	46
4.3.1.1	Erhebung	46
4.3.1.2	Auswertung	47
4.3.1.3	Visualisierung	47
4.3.2	Architektur	48
4.3.3	Technologie-Stack	50
4.3.4	Übertragbarkeit	51
4.4	Implementierung	52
5	Ergebnis	53
5.1	Demonstration	53
5.2	Kriterien	53
5.3	Übertragbarkeit	53
5.4	Einschätzung von anderen Entwicklern (optional)	53
6	Abschluss	55
6.1	Fazit	55
6.2	Ausblick	55
	Anhang	56
7	Anhang	56
7.1	Studien zur Browserkompatibilität	56
	Eidesstattliche Erklärung	57
	Abkürzungsverzeichnis	58
	Abbildungsverzeichnis	59
	Tabellenverzeichnis	60
	Quellcodeverzeichnis	60
	Literaturverzeichnis	61

1 Einleitung

1.1 Motivation

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

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

Während in der Vergangenheit Online-Auftritte i. d. R. als Webapplikation mit serverseitigen Rendering realisiert wurden, sind heutzutage Javascript-basierte Webapplikation mit clientseitigem Rendering die Norm. Bei einer solchen Webapplikation befindet sich die gesamte Logik mit Ausnahme der Berechnung des Angebots und der Verarbeitung der Antragsdaten im Browser des Nutzers.

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

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

1.2 Zielsetzung

Das grundlegende Ziel dieser Arbeit soll es sein, den Betreibern einer JavaScript-basierten Webapplikation die Möglichkeit zu geben das Verhalten ihrer Applikation und die Interaktionen von Nutzern. Diese Nachvollziehbarkeit soll insbesondere bei Fehlerfällen u. Ä. gewährleistet sein, aber auch in sonstigen Fällen soll eine Nachvollziehbarkeit möglich sein. Eine vollständige Überwachung der Applikation und des Nutzers (wie bspw. bei Werbe-Tracking) sind jedoch nicht vorgesehen. Daraus ergibt sich die Forschungsfrage:

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

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

Die anzustrebende Lösung soll ein Proof-of-Concept sein, welches anhand einer bestehenden Demoanwendung erstellt werden soll. Die Demoanwendung soll repräsentativ eine abgespeckte JavaScript-basierte Webapplikation darstellen, bei der die zuvor benannten Hürden zur Nachvollziehbarkeit bestehen.

Vor der eigentlichen Lösungserstellung soll jedoch die theoretische Seite beleuchtet werden, indem die Nachvollziehbarkeit sowie Methoden und Praktiken zur Erreichung dieser beschrieben werden. Es gilt aktuelle Literatur und den Stand der Technik zu erörtern, in Bezug auf die Forschungsfrage. Beim Stand der Technik sollen Technologien aus Fachpraxis und Literatur näher betrachtet werden und beschrieben werden.

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

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

1.2.1 Abgrenzung

Die Demoanwendung wird als Single-Page-Application (SPA) realisiert, denn hier bewegt sich das Projektumfeld von der Open Knowledge GmbH. Bei der Betrachtung der Datenerhebung und -verarbeitung ist eine volle Konformität mit der DSGVO nicht zu prüfen.

Bei der Erörterung zum Stand der Technik sollen detaillierte Produktvorstellungen nicht das Ziel sein, auch ist keine umfassende Gegenüberstellung anzustreben.

1.3 Vorgehensweise

Zur Vorbereitung eines Proof-of-Concepts wird zunächst die Ausgangssituation geschildert. Speziell wird auf die Herausforderungen der Umgebung „Browser“ eingegangen, besonders in Hinblick auf die Verständniserlangung zu Interaktionen eines Nutzers und des Verhaltens der Applikation. Des Weiteren wird die Nachvollziehbarkeit als solche formal beschrieben und was sie im Projektsfeld genau bedeutet.

Darauf aufbauend werden allgemeine Methoden vorgestellt, mit der die Betreiber und Entwickler eine bessere Nachvollziehbarkeit erreichen können. Dabei werden die Besonderheiten der Umgebung beachtet und es wird erläutert, wie diese Methoden in der Umgebung zum Einsatz kommen können. Hiernach sind Ansätze aus der Literatur und Fachpraxis zu erörtern, welche eine praktische Realisierung der zuvor vorgestellten Methoden darstellen.

Auf Basis des detaillierten Verständnisses der Problemstellung und der Methoden wird nun ein Proof-of-Concept erstellt. Ziel soll dabei sein, die Nachvollziehbarkeit einer Webapplikation zu verbessern. Der Proof-of-Concept erfolgt auf Basis einer Demoanwendung, die im Rahmen dieser Arbeit erstellt wird.

Ist ein Proof-of-Concept nun erstellt, wird analysiert, welchen Einfluss es auf die Nachvollziehbarkeit hat und ob die gewünschten Ziele erreicht wurden (vgl. Zielsetzung).

1.4 Open Knowledge GmbH

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

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

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

2 Ausgangssituation

2.1 Browserumgebung

Web Browser haben sich seit der Veröffentlichung von Mosaic, einer der ersten populären Browser, im Jahr 1993 stark weiterentwickelt. Das Abrufen und Anzeigen von statischen HTML-Dokumenten wurde mithilfe von JavaScript um interaktive und später dynamische Inhalte erweitert. Heutzutage können in Browsern komplexe Webapplikationen realisiert werden, welche zudem unabhängig von einem speziellen Browser entwickelt werden können. Durch diese Entwicklung und die breiten Anwendungsfälle, besitzt die Umgebung „Browser“ besondere Eigenschaften, welche nachfolgend beschrieben werden.

2.1.1 Browserprodukte

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

Generell lässt sich feststellen, dass auch in der Literatur die Veröffentlichungen in Bezug auf (In-)Kompatibilität von Browsern abnimmt, dies ist in Abbildung 2.1 zu betrachten. Dies spricht dafür, dass das Problem von XBIs nicht mehr so präsent ist, wie zuvor.

Im Jahr 2020 gab es weitere Entwicklungen, die die Kompatibilität zwischen Browsern erhöhte. Microsoft ist beim Folgeprodukt zum Internet

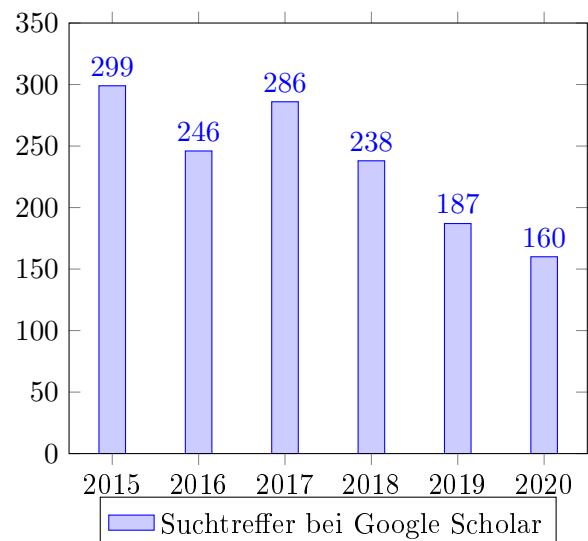


Abb. 2.1: Studien zur Browserkompatibilität, eigene Darstellung (vgl. 7.1)

Explorer, den Microsoft Edge Browser, von einer proprietären Browserengine zu Chromium gewechselt [Mic20b] und enthält somit denselben Kern wie Chrome und Opera. Zum Ende 2020 wird zudem der Support für den Internet Explorer 11 eingestellt [Mic20a]. Im September 2020 meldete Statista eine Marktverteilung von 69,71% Chrome, 8,73% Safari, 8,15% Firefox, 5,54% Edge, 2,51% Internet Explorer und 2,3% Opera [Sta20].

2.1.2 JavaScript

Als JavaScript 1997 veröffentlicht und in den NetScape Navigator integriert wurde, gab es die berechtigten 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 Sandbox. [Pow06]

Die JavaScript-Sandbox bei Browsern schränkt ein, dass unter anderem kein Zugriff auf das Dateisystem erfolgen kann. Auch Zugriff auf native Bibliotheken oder Ausführung von nativem Code ist nicht möglich [OKSK15]. Browser bieten dafür aber einige Schnittstellen an, die es erlauben z. B. Daten beim Client zu speichern oder auch Videos abzuspielen.

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

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

Über die Jahre wurde der JavaScript Standard immer umfangreicher und dies erlaubte Entwicklern mächtige Werkzeuge sowie Frameworks zu entwickeln, um u. A. Webapplikationen bereitzustellen. Webapplikationen konnten nun dazu verwendet werden, um einen großen Teil der Funktionalitäten eines Produktes abzubilden. Diese „clientbasierten“ Anwendungen werden im nächsten Abschnitt näher beleuchtet.

2.2 Clientbasierte Webapplikationen

2.2.1 JavaScript-basierte Webapplikationen

Eine JavaScript-basierte Webapplikation, ist eine Webapplikation, in der die Hauptfunktionalitäten über JavaScript realisiert werden. Dies umfasst unter anderem Interaktivität und dynamische Inhaltsdarstellung. Hierbei werden meist nur Grundgerüste in HTML und gegebenenfalls auch CSS bereitgestellt, die eigentlichen Inhalte werden dynamisch mit JavaScript erstellt. Die Inhalte werden überwiegend über zusätzliche Schnittstellen der Webapplikation bereitgestellt.

2.2.2 Single-Page-Applications

Single-Page-Applications (SPAs) sind eine Submenge der JavaScript-basierten Webapplikationen und gehen bei der dynamischen Inhaltsdarstellung einen Schritt weiter. Die gesamte Anwendung wird über ein einziges HTML-Dokument und die darin referenzierten Inhalte erzeugt. Auf Basis dessen wird oftmals viel der Logik im Clientteil angesiedelt, welches die Anwendung in einen Rich- bzw. Fat-Client verwandelt.

Für das Bereitstellen einer solchen Applikation, ist unter anderem nur ein simpler Webserver notwendig und ein oder mehrere Dienste, von dem aus die SPA ihre Inhalte abrufen kann. Populäre Frameworks, um SPAs zu Erstellen, sind beispielsweise Angular [Goo20a], React [Fac20b] oder Vue.js [YM20].

Neben der Eigenschaft eine Alternative zu anderen Webapplikationen zu sein, bieten SPAs zudem den Stakeholdern die Möglichkeit diese als Offline-Version zur Verfügung zu stellen. Grund dafür ist, dass eine SPA bereits jedwede Logik enthält, sind zusätzlich keine externen Daten notwendig, so kann eine SPA simpel als Offline-Version bereitgestellt werden. Weiterhin steigern SPAs die User Experience (UX), indem sie unter anderem schneller agieren, da keine kompletten Seitenaufrufe notwendig sind [AMWR20].

Durch diesen brachial anderen Ansatz, gibt es aber auch negative Eigenschaften. Unter anderem werden native Browserfunktionen umgangen, wie die automatisch befüllte Browserhistorie oder auch Aktionen zu Verlinkungen, wie Scrollrad-Klicks oder Lesezeichen, die mit „virtuelle“ Links und Buttons nicht möglich sind. Dies ist zu Teilen auch bei JavaScript-basierten Webapplikationen der Fall. Um diese Funktionalitäten wiederherzustellen sind für die zuvor genannten Frameworks Angular, React und auch Vue.js zusätzliche Bibliotheken notwendig.

Nichtsdestotrotz kann ein jahrelanger Trend von der Adaption von Single-Page-Applications erkannt werden und eine große Auswahl an erprobten Technologien stehen heutzutage zur Verfügung [GB20].

2.3 Nachvollziehbarkeit

Nachvollziehbarkeit bedeutet allgemein, dass über ein resultierendes Verhalten eines Systems auch interne Zustände nachvollzogen werden können. Dies ist keine neue Idee, sondern fand bereits 1960 im Gebiet der Kontrolltheorie starke Bedeutung [Ká60]. Nach Freedman [Fre91] und Scrocca *et al.* [STM+20a] lässt sich diese Definition auch auf Softwaresysteme übertragen.

In dieser Arbeit wird sich mit der Nachvollziehbarkeit in speziellen Situation befasst, nämlich wenn die Betreiber und Entwickler das Verhalten einer Webapplikation und die Interaktionen eines Nutzers verstehen möchten.

TODO: Definition von Observability von FAME einfügen

2.3.1 Nutzen

Tritt beispielsweise ein Softwarefehler (Bug) bei einem Nutzer auf, aber die Betreiber erhalten nicht ausreichende Informationen, so kann der Bug ignoriert werden oder gering priorisiert und in Vergessenheit geraten. Dies geschah im Jahr 2013, als Khalil Shreath eine Sicherheitslücke bei Facebook fand und diesen bei Facebooks Bug-Bounty-Projekt Whitehat meldete [SD13]. Sein Fehlerreport wurde aufgrund mangelnder Informationen abgelehnt:

Unfortunately your report [...] did not have enough technical information for us to take action on it. We cannot respond to reports which do not contain enough detail to allow us to reproduce an issue.

Durch den Bug konnte Shreath auf die private Profilseite von Nutzern schreiben, ohne dass er mit ihnen vernetzt war. Um Aufmerksamkeit auf das Sicherheitsproblem zu erregen, hinterließ er eine Nachricht auf Facebooks Gründer und CEO Mark Zuckerbergs Profilseite. Erst hiernach nahm sich Facebooks Team dem Problem an.

2.3.2 Nachvollziehbarkeit bei SPAs

Wie zuvor in Abschnitt 2.2 „Clientbasierte Webapplikationen“ geschildert, gibt es bei Webapplikationen und insbesondere Single-Page-Applications besondere Barrikaden, die es den Betreibern und Entwicklern erschwert das Verhalten einer Applikation und die Interaktionen eines Nutzers nachzuvollziehen.

Bei SPAs ist die Hauptursache, dass die eigentliche Applikation nur beim Client läuft und nur gelegentlich mit einem Backend kommuniziert.

2.3.3 Browserbedingte Hürden

2.3.3.1 Cross-Origin Resource Sharing (CORS)

Wie in der Geschichte zu JavaScript beschrieben, stellt CORS eine natürliche Entwicklung zur Erweiterung des Funktionsumfangs von JavaScript dar. Um nicht auf einen einzelnen Webserver beschränkt zu sein, wurde mit CORS diese Einschränkung eingegrenzt. Wichtiger Bestandteil von CORS ist aber eher die Absicherung vor Missbrauch, weswegen die Einschränkungen überhaupt existierte. Das Konzept von CORS stellt sicher, dass aus einer JavaScript-Umgebung heraus keine Ressourcen von Webservern angefragt werden, außer diese Webserver stimmen der Anfrage zu [MM20c].

In Abbildung 2.2 kann betrachtet werden, wie eine „cross-origin“ Ajax-Anfrage nach dem Konzept von CORS gehandhabt wird. Hierbei ist zu sehen, dass wenn der Aufruf nicht standardmäßig¹ ist, der Browser eine zusätzliche OPTIONS Anfrage an den jeweiligen Webserver sendet - dies stellt einen sogenannten „Preflighted Request“ dar. Wenn der Webserver in seiner Antwort auf die OPTIONS Anfrage nun bestätigt, dass die Methode, die Header und der „Content-Type“ so erlaubt sind, wird auch die eigentliche Ajax-Anfrage ausgeführt. Ansonsten schlägt die Anfrage fehl und im JavaScript-Kontext ist lediglich der Fehlschlag zu sehen, ohne einen Hinweis auf die Diskrepanz bzgl. CORS.



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

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

2.3.3.2 Content-Security-Policy

Eine Content-Security-Policy (CSP) definiert, welche Funktionalitäten einer Webapplikation aus geladen zur Verfügung stehen. Dies dient unter anderem dem Schutz vor Cross-Site-Scripting, indem eine Webapplikation beschränken kann, welche Funktionalitäten in JavaScript verfügbar sind und von wo aus JavaScript und CSS Skripte geladen werden dürfen [MM20b]. Weiterhin kann bei einem Versuch der Webapplikation die Regeln zu umgehen Bericht darüber erstattet werden.

2.3.4 Logdaten

Ähnlich wie bei anderen Umgebungen gibt es eine standardisierte Log- bzw. Konsolenausgabe für die JavaScript Umgebung [MM20a]. Diese Ausgabe ist aber für den Standard-Benutzer eher unbekannt und der Zugriff darauf sowie die Funktionen dessen können je nach Browser variieren. Deswegen kann in den meisten Fällen nicht darauf gehofft werden, dass die Nutzer dieses Log bereitstellen. Zusätzlich ist es durch die zuvor beschriebenen Härtungsmaßnahmen von Browsern nicht möglich das Log in eine Datei zu schreiben.

Eine Automatisierung der Logdatenerhebung ist zudem auch nicht trivial, denn die Daten, welche in die Ausgabe geschrieben werden, sind nicht aus der JavaScript Umgebung aus lesbar. Alternativ können die Daten selber erhoben oder abgefangen werden. Hierbei besteht aber weiterhin die Hürde, wie die Daten an ein Partnersystem gelangen.

2.3.5 Fernzugriff

Ein weiterer Punkt, der die Umgebung „Browser“ von anderen unterscheidet ist, dass die Betreiber und Entwickler sich normalerweise nicht auf die Systeme der Nutzer schalten können. Bei Experten Anwendungen ginge dies vielleicht, aber wenn eine Webapplikation für den offenen Markt geschaffen ist, sind die Nutzer zahlreich und unbekannt.

Weiterhin gibt es standardmäßig keine Funktionalität wie z. B. das Remote Application Debugging [Ora20], welches Java unterstützt.

3 Methoden und Praktiken

3.1 Methoden

3.1.1 Logging

Mit Logging bezeichnet man die systematische Protokollierung von Softwareprozessen und ihren internen Zuständen. Mithilfe von Logs, also Protokollen, kann Betreibern und Entwicklern ermöglicht werden, dass sie das Anwendungsverhalten nachvollziehen können. Beispielsweise ist es sinnvoll einen fehlgeschlagenen Login eines Nutzers zu loggen, dabei ist es hilfreich, wenn viele Kontextinformationen mitgeloggt werden.

Logging bei Webfrontends stellt eine besondere Hürde dar, denn wie bereits in Unterabschnitt 2.3.4 geschildert, müssen die Logs an ein Partnersystem weitergeleitet werden.

3.1.1.1 Structured Logging

Logmeldungen erfolgen meist textbasiert und in einem menschenlesbaren Format. Wenn nun jedoch Informationen aus einer großen Menge von Logs extrahiert werden sollen, ist so ein simples Format hinderlich. Hierbei kommt Structured Logging ins Spiel, bei dem zwar auch menschenlesbare Logmeldungen produziert werden, aber das Format ist fest definiert und erlaubt einer anderen Software Informationen einfacher zu extrahieren.

3.1.2 Metriken

Mit Metriken versucht man Softwareeigenschaften in Zahlenwerten abzubilden. Diese Zahlenwerte werden über Messungen ermittelt. Beispiele für Metriken sind:

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

Mit den Ergebnissen lassen sich wiederum Rückschlüsse zur Softwareeigenschaft ziehen, dass bspw. eine Anfrage deutlich länger benötigt als andere „gleichwertige“ Anfragen. Weiterhin lassen sich historische Veränderungen in den Metriken erkennen und können unerwünschte Abweichungen aufdecken.

3.1.3 Tracing

Tracing beschäftigt sich mit dem Aufzeichnen von Kommunikationsflüssen. Hierbei erfasst Tracing einerseits die Kommunikationsflüsse innerhalb einer Applikation oder innerhalb eines Systems. Andererseits zeichnet Tracing aber auch die Kommunikationsflüsse bei verteilten Systemen auf, um diese, meist komplexen Zusammenhänge, zu veranschaulichen. Ein herstellerunabhängiger Standard, der sich aus diesem Gebiet entwickelt hat, ist OpenTracing [Ope20g].

OpenTracing definiert zwei grundlegende Objekte: Traces und Spans. Ein Trace ist eine Menge an Events, die über eine einzelne logische Aktion - wie z. B. den Druck eines Knopfes - ausgelöst wurde oder resultiert. Spans besitzen den Namen einer Methode oder Operation, welche der Span umschließt. Wird in der Methode oder Operation eine weitere Methode oder Operation aufgerufen, welche von einem Span umschlossen ist, so ist dies nun ein Kindspan des ursprünglichen Spans. Spans haben einen Start- sowie einen Endzeitpunkt und sie speichern die Beziehung zu ihrem Elternspan (außer es handelt sich um den „root“-Span). Sie können darüber hinaus Attribute enthalten, sowie eingetretene Events.

OpenTracing definiert Traces implizit über ihre Spans. Ein Trace ist damit ein gerichteter Graph ohne Zyklus, wobei die Knoten hierbei die Spans darstellen und die Kanten die Eltern-/Kindbeziehung veranschaulichen [Ope20f]. Ein beispielhafter Trace mit seinen Spans und deren Beziehungen ist in Abbildung 3.1 zu betrachten. Die zeitliche Reihenfolge der Spans kann auch eine hilfreiche Visualisierung sein, um die Entwicklung der Spans zu verstehen [cite] (vgl. Abbildung 3.2). Diese Definition wurde in den Nachfolgestandard OpenTelemetry aufgenommen und an späterer Stelle erläutert.

Ein verteilter Trace, oftmals „Distributed Trace“ genannt, ist ein Trace, der sich aus den Events von verschiedenen Systemen zusammensetzt, die miteinander kommunizieren. Hierbei beinhaltet der Trace Events, die über die Grenzen von Anwendungen, Prozessen und Netzwerken hinausgehen [Ope20f].



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



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

3.1.4 Fehlerberichte

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

Bettenburg *et al.* [BJS⁺08] fanden jedoch Mängel bei der Effektivität von Fehlerberichten. Denn Nutzer meldeten Informationen und Details, die sich für die Entwickler als nicht allzu hilfreich herausstellten. Diese Diskrepanz kann u. A. dadurch erläutert werden, dass Nutzer im Regelfall kein technisches Verständnis vom System vorweisen.

Hinzukommend muss man beachten, dass die User Experience negativ beeinflusst werden kann, wenn ein Nutzer einen Dialog mit einem Aufruf für einen Fehlerbericht erhält - gerade wenn der Fehler ihn selbst nicht eingeschränkt hat.



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

3.2 Praktiken aus der Wirtschaft

In der Fachpraxis haben sich einige Technologien über die Jahre entwickelt und etabliert, die eine verbesserte Nachvollziehbarkeit als Ziel haben. Neben den zuvor vorgestellten Methoden haben sich bei Technologien einige Praktiken entwickelt, die teilweise auf den Methoden aufbauen oder neue Konzepte darstellen, um ihr Ziel zu erreichen. Diese Praktiken werden folgend vorgestellt.

3.2.1 Observability

Observability ist der Oberbegriff, der sich in der Wirtschaft entwickelt hat, um Werkzeuge zu beschreiben, die eine bessere Nachvollziehbarkeit versprechen. Es kann mit „Beobachtbarkeit“ übersetzt werden, aber ließe sich auch freier mit Nachvollziehbarkeit gleichsetzen. Observability entstand aus dem klassischen Monitoring von Software, aber beinhaltet neben Monitoring noch einige weitere Disziplinen, wie Logging, Tracing und Metriken. Graf [Gra20] definiert Observability wie folgt:

Ziel ist es, Anwendungen und Systeme exakt zu beobachten, Informationen über technische Fehlfunktionen zu erhalten und schnell an die Verantwortlichen zu übermitteln. Logfiles, Informationen zur Ressourcennutzung und Anwendungs-Traces sollen den Administratoren und Entwicklern dabei helfen, die Fehlerursachen zu erkennen, die Probleme zu beheben und künftige Ausfälle nach Möglichkeit zu vermeiden [Sri18].

3.2.2 System Monitoring

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

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

3.2.3 Log Management

Log Management umfasst die Erfassung, Speicherung, Verarbeitung und Analyse von Logdaten von Anwendungen. Weiterhin bieten Werkzeuge hierbei oftmals fundierte Such- und Meldefunktionen. Um die Daten aus einer Anwendung heraus zu exportieren, werden oftmals eine Vielzahl an Integrationen für Frameworks und Logbibliotheken angeboten.

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

3.2.4 Application Performance Monitoring (APM)

Beim Application Performance Monitoring werden Daten innerhalb von Applikationen gesammelt, die Rückschlüsse auf die Performanz von bspw. Transaktionen geben sollen [ABC⁺16]. Mit diesen Daten können dann Regressionen der Performanz, in Aspekten wie Zeitaufwand oder Ressourcennutzung, festgestellt werden.

Neben allgemeinen Aspekten, die sich auf den Prozess- oder Ausführungskontext beziehen, können aber auch spezifische Faktoren, wie die Ausführungszeit einer wichtigen Methode, veranschaulicht werden. APM wird zum Großteil mit Metriken realisiert, welche die zu untersuchenden Aspekte in numerischen Werten wiedergeben.

3.2.5 Real User Monitoring (RUM)

Real User Monitoring beschäftigt sich mit dem Mitschneiden von allen Nutzerinteraktionen mit bspw. einer Webapplikation. Hiermit lässt sich nachvollziehen, wie ein Nutzer die Anwendung verwendet. RUM kann dazu verwendet werden um Herauszufinden, wie ein Nutzer zu einem Zustand gelangt ist. Aber es können auch ineffiziente Klickpfade hierdurch festgestellt werden und darauf basierend UX Verbesserungen vorgenommen werden.

Häufig wird RUM aber nicht verwendet, um einzelne Nutzerinteraktionen nachvollziehbar zu machen, sondern es werden oft gruppierte Verhaltensweisen erfasst, um allgemeine Schlüsse über eine Anwendung ziehen zu können.

3.2.6 Synthetic Monitoring

Beim Synthetic Monitoring werden Endnutzerszenarien simuliert, um zu prüfen und sicherzustellen, dass diese Szenarien wie gewünscht ablaufen. Hierbei kann auf Aspekte wie Funktionalität, Verfügbarkeit und auch verstrichene Zeit kontrolliert werden.

3.2.7 Error/Crash Monitoring

Das Error Monitoring konzentriert sich auf das Erfassen und Melden von Fehlern. Es werden oftmals neben dem eigentlichen Fehler auch Aspekte vom RUM und Logging gemeldet, um mehr Kontextinformationen zu liefern.

Das Error Monitoring wird oftmals eng mit einem Issue-Management verbunden, da aufgetretene Fehler meist zu einem Issue werden oder werden sollten und eine Nachverfolgbarkeit der Fehlerbehebung hierbei Sinn macht.

3.2.8 Session-Replay

Session-Replay bedeutet, dass eine Sitzung eines Nutzers nachgestellt wird, so als ob sie gerade passiert. Hierbei können einzelne Aspekte der Anwendung nachgestellt werden, bspw. der Kommunikationsablauf, bei dem die tatsächliche zeitliche Abfolge von Kommunikationen nachvollzogen werden können. Desto mehr Aspekte nachgestellt werden, desto realitätsnaher ist die Simulation und entsprechend hilfreich ist sie beim Nachvollziehen.

Session-Replay nimmt so eine enorme Datenmenge für jede Nutzersitzung auf und benötigt so ein gutes Konzept im Datenmanagement, aber besonders bei Browsern wird eine effiziente Kommunikation benötigt.

3.3 Werkzeuge und Technologien

3.3.1 Fachpraxis

Bei der Recherche zu Werkzeugen und Technologien aus der Fachpraxis wurden einige Produkte näher betrachtet. Folgend werden die Ergebnisse beschrieben, welche aber nicht als Vergleich oder Produktbeschreibung dienen, sondern eher, um ein grobes Verständnis der Fachpraxis zu vermitteln, um im Verlauf der Arbeit darauf zurückgreifen zu können. Bei der Recherche kam wiederholt der Begriff OpenTelemetry auf, deswegen und aufgrund der Beziehung zu OpenTracing wird der Standard folgend kurz beschrieben.

3.3.1.1 OpenTelemetry

OpenTelemetry (OTel) [Ope20b] ist ein sich derzeit entwickelnder herstellerunabhängiger Standard, um Tracing-, Metrik- und Logdaten¹ zu erfassen, verarbeiten, analysieren und zu visualisieren. Der Standard fasst die beiden Standards OpenTracing und OpenCensus [Ope20a] zusammen und hat sich als Ziel gesetzt diese zu erweitern [Jos19]. Hinter dem Standard stehen u. A. die Cloud Native Computing Foundation (CNCF), Google, Microsoft, und führende Hersteller von Tracing- und Monitoring-Lösungen. Ein erster Release ist für Ende 2020/Anfang 2021 geplant. Ziel ist es, dass Entwicklertools und -werkzeuge benutzen können, ohne jedesmal eine hochspezifische Anbindung schreiben und konfigurieren zu müssen. Stattdessen definiert der Standard Komponenten, die spezielle Aufgabengebiete haben und mit einer allgemeinen API anzusprechen sind. Die technische Infrastruktur einer auf OTEL basierenden Lösung ist in Abbildung 3.4 zu betrachten. Im groben definiert OTel folgende Komponenten: API, SDK, Exporter, Collector und Backend (vgl. Abbildung 3.5).



Abb. 3.4: Schaubild einer Lösung auf Basis von OTel [Ope20c]



Abb. 3.5: OTel Komponenten [Dyn20c]

¹Logging ist noch nicht gut im OTel Standard definiert/aufgenommen [Ope20e]

3.3.1.2 New Relic

New Relic [New20b] ist ein SaaS der gleichnamigen Firma, welcher Betreiber von Softwareprojekten dabei unterstützt das Verhalten ihrer Anwendungen zu überwachen. Der Dienst konzentriert sich auf System Monitoring, APM und RUM und erfasst die notwendigen Daten mit proprietären Lösungen. Neben den Kernfunktionalitäten unterstützt New Relic auch Log Management, Synthetic Monitoring, Tracing und Error Monitoring.

Bei New Relic wurde die kostenlose Version aufgesetzt und evaluiert. Der New Relic Agent, welcher die Daten beim Client sammelt, wird über ein Skript eingebunden und sendet in regelmäßigen Abständen Daten an New Relic. Über die Oberfläche von New Relic können dann allgemeine Charakteristika der Clients betrachtet werden, wie Ladezeiten, Browserhersteller, Ajax-Antwortzeiten. Spezielle Eigenschaften eines einzelnen sind jedoch nicht möglich, wahrscheinlich u. A. aus Aspekten des Datenschutzes. Jedoch im Fehlerfall gibt es mehr Informationen, denn hier werden spezifische Daten (Stacktrace, genaue Browserversion, Uhrzeit, ...) zum Fehler sowie zum Client erhoben und in der Oberfläche dargestellt. Diese Informationen sind gut zum Nachvollziehen, dass ein Fehler aufgetreten ist und was die Rahmenbedingungen sind - jedoch sind die Informationen nicht ausreichend, um ein klares Bild des Fehlers zu erhalten. Dies könnte mit dem Log-Management von New Relic komplementiert werden, jedoch wären diese selber zu erheben und an einen eigenen Server zu senden, um dann die Logs an New Relic weiterzuleiten. Des Weiteren wird der Agent standardmäßig von AdBlockern blockiert, sowie von der „Enhanced Tracking Protection“ im Mozilla Firefox. Es ist nicht möglich New Relics Dienst selber zu hosten, es ist also eine sog. „OnPremise“-Lösung nicht möglich. Des Weiteren wird auch eine Weiterleitung über einen eigens bereitgestellten Proxy nicht unterstützt.

New Relic gibt an, dass Daten nach dem OpenTelemetry Standard selber erfasst und an New Relic gesendet werden können, ohne eine proprietäre Software [New20a]. Leider ist dieses Feature in der Testversion, die zum Evaluieren benutzt wurde, nicht enthalten und kann somit nicht bestätigt werden. Es sind jedoch offizielle und quelloffen veröffentlichte Exporter für New Relic verfügbar für .NET, Python und Java [Ope20d].

3.3.1.3 Dynatrace

Dynatrace [Dyn20b] ist ein SaaS des gleichnamigen Unternehmens, welcher Betreiber von Softwareprojekten dabei unterstützt das Verhalten ihrer Anwendungen zu überwachen. Der Dienst konzentriert sich auf System Monitoring, APM und RUM und erfasst die notwendigen Daten mit proprietären Lösungen, dem „OneAgent“. Ganz ähnlich wie New Relic unterstützt Dynatrace neben den Kernfunktionalitäten auch Log Management, Synthetic Monitoring, Tracing und Error Monitoring.

Bei Dynatrace wurde die 14-tägige Testversion aufgesetzt und evaluiert. Wie zuvor genannt, erfolgt die Datenerhebung über den Dynatrace OneAgent, welcher genauso wie New Relics Agent kontinuierlich Daten sendet. Die Oberfläche von Dynatrace stellt auch ungefähr dieselben Informationen dar wie New Relic, wobei Dynatrace das Ganze visuell ansprechender darstellt. Dynatrace bietet zudem auch die Funktionalität des Error Monitoring, aber leidet unter demselben Problem wie New Relic: zu wenig Kontextinformationen.

Der Dynatrace OneAgent, wird standardmäßig von AdBlockern blockiert, aber nicht wie New Relic von der „Enhanced Tracking Protection“ des Mozilla Firefox. Dynatrace kann zudem damit punkten, dass es als OnPremise-Lösung verfügbar ist, sodass Kunden genau bestimmen können, wo die Daten verarbeitet und gespeichert werden.

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

3.3.1.4 Sentry

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

Für Sentry wurde die kostenlose Version aufgesetzt und evaluiert. Sentry bietet bei NPM quelloffene Pakete an [Sen20a], um Fehler zu erfassen und an Sentry zu melden. Es werden Pakete für folgende Frameworks bereitgestellt: JavaScript, Angular, AngularJS, Backbone, Ember, Gatsby, React und Vue. Das Aufsetzen stellt sich einfach dar und ermöglicht einige Konfigurations- und Verarbeitungsoptionen, bspw. können sensible Daten aus den Datenpaketen entfernt werden oder weitere Informationen gemeldet werden. Anders als bei den beiden vorherigen Tools wird zu Sentry nur kommuniziert, wenn ein Fehler auftritt. Hierbei werden dafür aber umso mehr Daten erhoben: Detaillierte Klickpfade des Nutzers, Logmeldungen der Browserkonsole sowie die Informationen, die auch die anderen Tools bereitstellen.

Gemeldete Fehler werden in „Issues“ umgewandelt, welche einem Fehlerticket entsprechen und in der Oberfläche Funktionen zum Zuweisen und zum Nachhalten der Behebung bieten. Sentry versucht die eingetroffenen Fehler in bestehenden Issues zu gruppieren, sodass jeweils zusammengehörige Fehler auch zusammen behandelt werden. Bei Sentry fehlt jedoch die ganzheitliche Nachvollziehbarkeit, also das man auch nicht im Fehlerfall nachvollziehen kann, was geschehen ist. Dafür sind die gesammelten Fehlerinformationen zahlreich und helfen beim Nachvollziehen besser als die vorhergegangenen Werkzeuge.

Der Quellcode von Sentry wurde veröffentlicht und weiterhin wird bei Sentry auch eine OnPremise-Lösung, basierend auf Docker, angeboten [Sen20b].

3.3.1.5 LogRocket

LogRocket [Log20a] ist ein SaaS Produkt des gleichnamigen Unternehmens und konzentriert sich auf detailliertes Session-Replay von JavaScript-basierten Clientanwendungen, um Probleme zu identifizieren, zu nachvollziehen und lösen zu können. Session-Replay ist auch die einzig identifizierbare Kernfunktionalität, die LogRocket aufweist.

Für die Evaluierung wurde die kostenlose Testversion von LogRocket verwendet. Zur Datenerhebung wird das Paket `logrocket` von NPM hinzugezogen und nach der Initialisierung sammelt es eigenständig die notwendigen Daten. Mithilfe der gesammelten Daten wird die gesamte Sitzung des Nutzers nachgestellt, hierbei ist die Anwendung, die Nutzerinteraktionen, die Netzwerkaufrufe sowie das DOM zu sehen. Die Nachstellung wird videoähnlich aufbereitet und erlaubt ein präzises Nachvollziehen der zeitlichen Reihenfolge und Bedeutung (vgl. Abbildung 3.6).

Neben dem JavaScript SDK bietet LogRocket quelloffene Plugins für folgende Bibliotheken: Redux, React, MobX, Vuex, ngRx, React Native. LogRocket ist zudem als OnPremise-Lösung verfügbar. Zusätzlich bietet LogRocket auch einige Integrationen für andere Tools, wie z. B. Sentry. Die Integration mit Sentry wurde ebenso evaluiert, hierdurch wurde ermöglicht, dass von einem Sentry Issue direkt auf das Session-Replay des konkreten Fehlerfalls in LogRocket gesprungen werden konnte.



Abb. 3.6: Beispiel eines Session-Replays bei LogRocket

3.3.1.6 Splunk

Splunk [Spl20] ist ein Softwareprodukt und ein SaaS des gleichnamigen Unternehmens. Splunk fungiert als eine universelle Datensinke, z. B. für Metriken und Logdaten. Es

bietet Funktionen diese Daten zu durchsuchen, überwachen, analysieren und visualisieren. Weiterhin wird Splunk klassisch für Log Management eingesetzt, kann aber auch Aspekte von APM, RUM und Error Monitoring erfüllen.

Splunks kostenlose Version der SaaS und der OnPremise-Lösung wurden evaluiert. Hierbei wurde zunächst festgestellt, dass das Daten senden vom Browserkontext zu Splunk nicht direkt möglich ist, da Splunk mit ablehnenden CORS-Headern antwortet. Hierfür wurde ein extra Dienst geschrieben, der die Daten vom Frontend annimmt und an Splunk weitersendet. Weiterhin konnten hierdurch Daten wie den User-Agent und die IP vom Client ermittelt und an die Datensätze angehängen werden.

An Splunk wurden über dieselbe Schnittstelle Log- und Errorinformationen gesendet. Hierbei konnte Splunk gut das Logmanagement erfüllen, denn Logeinträge ließen sich einfach filtern und auch zu Graphen aufbereiten. Weiterhin ist das Error Monitoring auch zufriedenstellend und enthält alle Daten, die auch bspw. bei Sentry vorhanden sind - hierbei fehlt jedoch ein Issue-Management, welches Sentry bereitstellt. Lediglich Tracing lässt sich nicht zufriedenstellend mit Splunk realisieren. Denn hier können lediglich die Daten konsumiert werden, aber eine verständliche Darstellung mittels Trace-Gantt-Diagrammen fehlt gänzlich. Hiermit lassen sich keine spezialisierten Tracing-Werkzeuge ersetzen.

3.3.1.7 Honeycomb

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

Honeycomb wurde mit der kostenlosen Version evaluiert. Honeycomb bietet sog. „Beelines“ an, welche Werkzeuge zur automatischen Datenerfassung sind. Diese Beelines sind aber nur für Node.js, Go, Python, Java, Ruby und Rails verfügbar, aber nicht JavaScript im Browser. Deshalb wurden zur Evaluierung Werkzeuge von OpenTelemetry hinzugezogen und zusätzlich musste nur noch ein Exporter für Honeycomb geschrieben werden. Honeycomb wirkt komplett anders als New Relic und Dynatrace, denn es ist nicht ausgefertigt mit konkreten Graphen und Darstellungen, sondern bietet eine Oberfläche mit dem man auf Basis der Daten selber Abfragen, Graphen und andere Darstellungen selber erstellen kann. Auch ist es nicht mit Jaeger, welches gleich näher betrachtet wird, vergleichbar, denn es bietet deutlich mehr Möglichkeiten als striktes Tracing.

3.3.1.8 Jaeger

Jaeger wurde 2017 als ein Projekt der CNCF gestartet [Jae20b]. Es ist ein System für verteiltes Tracing und bietet Funktionalitäten zur Datensammlung, -verarbeitung, -speicherung bis hin zur Visualisierung. Jaeger unterstützt und implementiert den Standard OpenTracing und erlaubt auch das Konsumieren von Zipkin Daten, einem Konkurrenzprodukt. Eine Unterstützung des OpenTelemetry Standards ist derzeit im Gange. Weiterhin kann Jaeger dazu benutzt werden, Metriken nach Prometheus [Pro20] zu exportieren, einem weiteren CNCF Projekt zur Speicherung und Visualisierung von Daten.

Jaeger wurde nicht aufgesetzt, sondern über einen Arbeitskollegen bei der Open Knowledge veranschaulicht und darauf basierend evaluiert. Wie zuvor bei Honeycomb beschrieben, beschränkt sich Jaeger auf Tracing, aber spezialisiert sich dafür auf dieses Gebiet und bietet eine gesamte Infrastruktur zur Tracespeicherung und -analyse. Die Traces sind als Gantt-Diagramme dargestellt, wie in Abbildung 3.7 zu sehen ist. Hierbei sind sowohl hierarchische als auch zeitliche Beziehungen visualisiert.

Anhand der Traces generiert Jaeger zudem automatisch eine Architektur, indem die Beziehungen zwischen Diensten zu sehen ist. In Abbildung 3.8 kann so eine Darstellung betrachtet werden, hierbei sieht mehrere Microservices und Datenbanken.

3.3.1.9 Weiteres

Bei der Recherche und Evaluierung wurden nicht alle auf dem Markt verfügbaren Werkzeuge und Technologien tiefergehend betrachtet. Deshalb werden weitere Funde, die nicht betrachtet wurden, hier kurz notiert:

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

Auch diese Auflistung stellt nicht die komplette Bandbreite an Werkzeugen und Technologien dar und die vorhergehende Betrachtung ist nicht als Empfehlung zu verstehen.

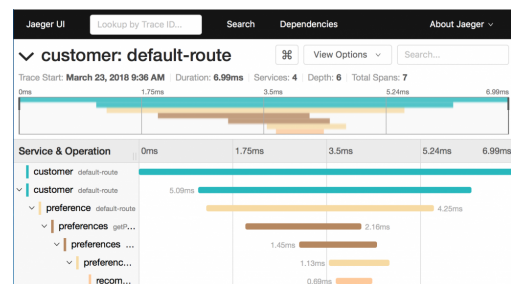


Abb. 3.7: Trace-Detailansicht. Quelle: Don Schenck [Sch20]

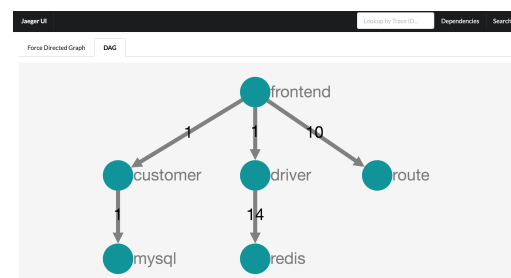


Abb. 3.8: Dienst-Abhängigkeits-Graph. Quelle: Yuri Shkuro [Shk20]

3.3.2 Literatur

In diesem Abschnitt sind die Ergebnisse aus der Literaturrecherche zusammengetragen: Verwandte Publikationen, die auf eine verbesserte Nachvollziehbarkeit abzielen.

3.3.2.1 TraVista

Ein erstes Werkzeug, welches in der Literatur identifiziert wurde, ist TraVista von Anand *et al.* [ASD⁺20]. Es basiert auf einem vorhergehenden Projekt von Anand, TraViz, welches nachgehend erläutert wird. TraVista beschäftigt sich mit dem Visualisieren von Tracedaten. Hierbei setzen sie auf den bestehenden Ansatz der Gantt-Diagramme auf, um Traces zu visualisieren. Die Gantt-Diagramme wurden jedoch um einige zusätzliche Informationen erweitert, wie in Abbildung 3.10 zu sehen ist.

Innerhalb eines Spans sind aufgetretene Events dargestellt, standardmäßig als weißer Balken oder bei selten auftretenden Events als schwarzer Balken (vgl. ⑦ in Abbildung 3.9). Beziehungen ⑥ zwischen den Spans werden visualisiert. Bei ② sind Histogramme zu sehen, die aus den gesammelten Metriken innerhalb der Traces, erstellt sind. Die Metrikwerte des aktuellen Trace sind blau hervorgehoben und erlauben hier auch eine visuelle Probe auf Ausreißer.



Abb. 3.9: Zoom von Abbildung 3.10, Abbildung aus [ASD⁺20]

Es lässt sich abschließend sagen, dass TraVista eine vielversprechende Visualisierung bietet. Jedoch wurde ein eigenes simples Backend entwickelt, welches nur mit einer vergleichsweise geringen Datenmenge getestet wurde (22.286 Traces, 147.812 Spans). Die Autoren selbst sehen Probleme bei der Skalierung, aber nannten auch, dass sie TraVista weiterentwickeln wollen, um ein neues Backend zu entwickeln.

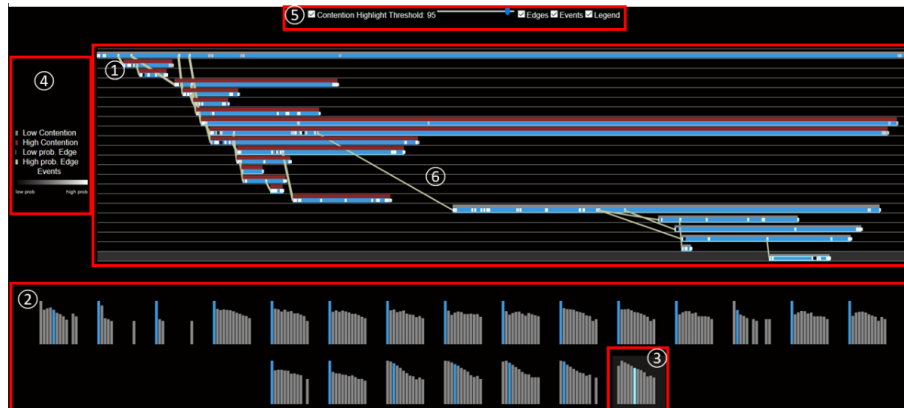


Abb. 3.10: TraVistas Gantt-Diagramm, Abbildung aus [ASD⁺20]

Vor der Arbeit an TraVista hat Vastaav Anand mit Matheus Stolet an einem weiteren Visualisierungstool für Traces gearbeitet: TraViz [AS20]. Es ist auf GitHub veröffentlicht <https://github.com/vaastav/TraViz>. TraViz bildet das Grundgerüst für die spätere Arbeit an TraVista und ist zudem das Backend für TraVista. Im Genaueren bedeutet dies, dass TraVista ein neuer Visualisierungsansatz für das Backend von TraViz ist. Die meisten funktionalen Eigenschaften von TraVista lassen sich ebenso auf das Backend von TraViz zurückverfolgen.

Mit Joseph Wonsil hat Anand an einem weiteren Tool rund ums Tracing gearbeitet: Tracey [AW20]. Es ist ebenso auf GitHub veröffentlicht <https://github.com/vaastav/Tracey>. Es beschäftigt sich mit der Aggregation von Tracingdaten, denn derzeitige Technologien beschäftigen sich hauptsächlich mit der Darstellung von einem einzelnen Trace. In diesem Bericht sind erneut Grundbausteine zu erkennen, die Anand wahrscheinlich bei TraVista anwenden konnte.

Sowohl TraViz (und TraVista) als auch Tracey werden jedoch seit Mai 2020 nicht mehr aktualisiert. Eine Nutzerschaft konnte auch nicht gefunden werden, da auf GitHub beide Projekte Null Mal favorisiert und Null Mal geklont wurden. Auch Issues oder Pull-Requests sind nicht vorhanden. Aufgrund dieser fehlenden Weiterentwicklung und den Skalierungsproblemen des Backends ist TraVista für diese Arbeit nicht von Relevanz.

3.3.2.2 „Bedeutung von Telemetry für den SDLC“

Neben TraVista wurde die im 2. Quartal 2020 erstellte Bachelorarbeit von Michael Graf bei der Recherche gefunden [Gra20]. Grafts Arbeit handelt über die „Bedeutung von Telemetry für den Software Development Cycle“. In der Arbeit beschäftigt er sich mit der

Software-Telemetrie als solche, aber legt den aktuellen Stand am Projekt OpenTelemetry dar. OpenTelemetry erhält enorme Bedeutung in der Arbeit und wird von Grund auf erläutert, betrachtet und analysiert.

Die Arbeit beschäftigt sich zunächst mit SDLC und wie moderne Ansätze hierfür Aussehen. Speziell wird der DevOps Ansatz beschrieben, bei dem eine kontinuierliche Weiterentwicklung des Produktes im Vordergrund steht. Hierbei bewertet Graf die Fähigkeit von Observability-Ansätzen nach ihrem tatsächlichen Nutzen und ihrer Übertragbarkeit. Dies erfolgt, wie zuvor beschrieben, am Beispiel von OpenTelemetry. Evaluert wird das Ganze durch das Aufsetzen von Jaeger mit einer Demoanwendung, durch das der Tracingaspekt von Observability realisiert wird.

Abschließend bewertet Graf OpenTelemetry positiv in den Punkten Interoperabilität, Plattformunabhängigkeit und Erweiterbarkeit. Jedoch mahnt er zudem, dass das Projekt noch jung ist und darauf basierend keine Prognosen über den Erfolg gemacht werden können. Graf sieht zudem Observability-Daten als eine gute Möglichkeit, um die Qualitätssicherung und -verbesserung zu unterstützen.

Grafs Arbeit dient somit als Vorbereitung für einen Lösungsansatz mit OpenTelemetry, bietet aber selbst keine neuen Ansätze.

3.3.2.3 Timelapse

Burg *et al.* [BBKE13] entwickelten 2013 eine neue Software, die es erlaubt das Anwendungsverhalten von Webapplikationen aufzunehmen und wieder abzuspielen: „Timelapse“. Timelapse erlaubt es direkt im selben Browser Nutzersitzungen aufzunehmen und wieder abzuspielen, sodass die simulierten Aktionen im Browser geschehen. Dies bedeutet, dass Entwickler und Betreiber gleichzeitig den DOM inspizieren können und auch Breakpoints im JavaScript-Code verwenden können.

Das Grundprinzip von Timelapse basiert darauf, dass die Browserumgebung gekapselt ist; sie lässt sich von außen nur über vordefinierte Schnittstellen ansprechen. Intern arbeitet die Browserumgebung mit einem einzigen Thread indem eine Eventloop die Logik ausführt. Um nun die Aufnahme-/Abspielfunktionalität zu ermöglichen wird ein eigens entwickeltes Framework hinzugezogen, „Dolos“ genannt.

Dolos schneidet interne DOM-Events sowie die Antworten von äußeren Schnittstellen mit (vgl. Abbildung 3.11). Beim Abspielen werden DOM-Events und

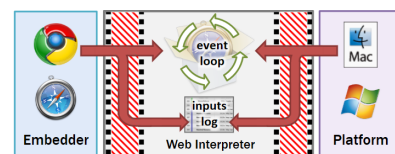


Abb. 3.11: Mitschneiden von DOM-Events, Abb. aus [BBKE13]

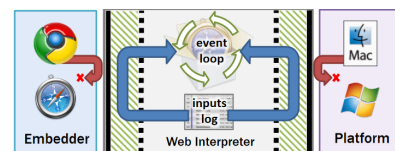


Abb. 3.12: Abspielen von DOM-Events, Abb. aus [BBKE13]

äußere Aufrufe in derselben Reihenfolge der Event-loop übergeben (vgl. Abbildung 3.12). Hierbei wird die Eventloop von anderen Einflüssen abgeschottet, so dass eine exakte Rekonstruktion erfolgen kann. Bei der Rekonstruktion ist es von enormer Bedeutung nicht-deterministische Eingaben korrekt zu simulieren, also Tastatureingaben, Netzwerkanfragen aber auch interne Ressourcen wie Zufallswerte und die aktuelle Zeit.

Das Projekt ist sehr vielversprechend, besitzt jedoch einige Eigenschaften, die es mit der in dieser Arbeit angestrebten Lösung inkompatibel machen. Es benötigt eine tiefe Integration in den Browser, speziell wird eine gepatchte Version von WebKit benötigt. Dies ist einerseits aufwändig, aber in diesem Projektumfeld quasi unmöglich, da die Nutzer als vielzählig und unbekannt gelten und genau diese Nutzersitzungen sollten nachvollziehbar gemacht werden. Weiterhin unterstützt das Projekt nur den Safari und Google Chrome Browser, wahrscheinlich aber auch nicht die aktuellen, denn es wird seit mehr als 5 Jahren nicht mehr weiterentwickelt. Der Quellcode findet sich jedoch auf GitHub <https://github.com/burg/replay-staging/>. Insgesamt ist das Projekt somit ungeeignet für eine die angestrebte Lösung.

3.3.2.4 FAME

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

Mit FAME wurde erforscht, ob die kombinierten Daten aus Nutzerfeedback und Monitoring dabei helfen können, um neue Anforderungen zu ergründen. Nach den Autoren ist dieser Ansatz neu, denn in der Literatur wurde zuvor hauptsächlich auf das Nutzerfeedback gesetzt, um neue Anforderungen zu identifizieren. Dies ist aber von Problemen geplagt, wie bspw. dem Verständnis des Problems oder auch wie relevant dieses Problem für die gesamte Nutzerschaft ist. Mithilfe von der Addition von Monitoring sollen einige dieser Einschränkungen begegnet werden, indem bspw. Funktionsgebiete über Metriken abgebildet werden und wenn ein negatives Feedback eines Nutzers zu diesem Gebiet aufkommt, kann in der Metrik abgelesen werden, ob das Verhalten von anderen Nutzern sich geändert hat. Eine abstrakte Übersicht des FAME-Frameworks kann in Abbildung 3.13 betrachtet werden.

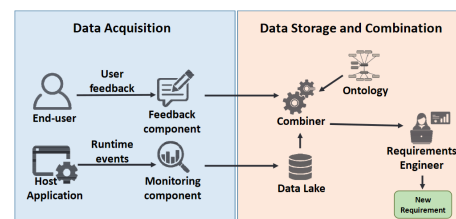


Abb. 3.13: Übersicht von FAME, Abb. aus [OS⁺18]

Das FAME-Framework definiert detailliert Komponenten, wie die Disziplinen des Monitoring und der Feedbackeinholung gelöst werden können, und wie dies in einer möglichst modularen Infrastruktur umgesetzt werden kann. Für die Kombination der Monitoring- und Feedbackdaten werden zudem Mechanismen geschaffen, dass effiziente Queries diese beide zusammen verbunden ausgeben. Diese Ansätze wurden mit der SEnerCon GmbH in einem Workshop auf die Probe gestellt und konnten dem Requirements-Engineer dabei unterstützen, neue Anforderungen zu identifizieren und irrelevante Anforderungen auszuschließen. Es konnte ein Mehrwert für die SEnerCon GmbH identifiziert werden.

Das Framework FAME gehört zum größeren Projekt SUPERSEDE, welches von der EU unterstützt wird [SUP20]. Das Framework ist quelloffen unter GitHub verfügbar: https://github.com/supersede-project/monitor_feedback. Seit Ende 2018 wird es jedoch *de facto* nicht mehr gewartet.

Das Projekt wird jedoch nicht bei der angestrebten Lösung verwendet. Zum Einen, weil das Ziel des Projektes nicht direkt auf Nachvollziehbarkeit von Nutzerinteraktionen und Anwendungsverhalten abzielt. Zu Anderem aber auch aufgrund der fehlenden Weiterentwicklung und Wartung. Dennoch sind die Erkenntnisse, dass eine Kombination aus unterschiedlichen Daten neue Sichtweisen ermöglicht, vielversprechend.

3.3.2.5 The Kaiju Project

Scrocca *et al.* [STM⁺20b] identifizierten 2020 eine Lücke im Gebiet der Observability, nämlich fehlt ein Werkzeug, welches die unterschiedlichen Datentypen Logs, Traces und Metriken kombiniert sammeln und aufbereiten kann und dies in quasi-Echtzeit. Aus diesem Grund wurde ein Konzept und ein Prototyp entwickelt, die darauf abzielen diese Lücke auszufüllen. Zunächst wird jedoch das Forschungsfeld und die Fachpraxis rund um die Observability aufgearbeitet. Hierbei definieren Scrocca *et al.* zwei Forschungsfragen, die es im Bericht zu beantworten gilt:

- Wie können wir das Systemverhalten enthüllen?
- Wie können wir Sinn aus dem Systemverhalten gewinnen?

Um diese Forschungsfragen zu beantworten, wurde ein Konzept erstellt, wie die Datentypen Logs, Traces und Metriken zusammen erhoben und kombiniert analysiert werden können. Es basiert auf einer allgemeinen Datenverarbeitung mittels einem Event-Stream-Processor

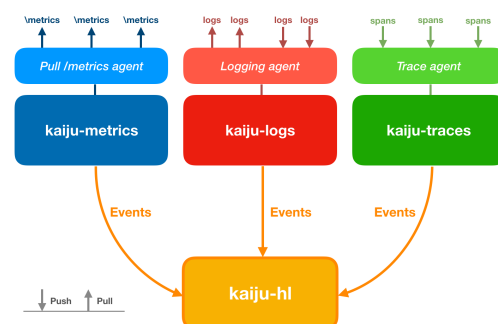


Abb. 3.14: Kaiju Architektur, Abb. aus [STM⁺20b]

(ESP). Das Kernmodul stellt somit das „kaiju-hl“ dar, welches in der Architektur in Abbildung 3.14 zu betrachten ist. Das Hauptmodul besteht selber aus mehreren Subkomponenten, die spezielle analytische Aufgaben übernehmen. Der Prototyp wurde mit der quelloffenen ESP-Engine Esper [Esp21] umgesetzt.

Aus den analysierten Daten erstellt Kaiju angereicherte und in der Struktur homogenisierte Daten, sogenannte High-Level-Events. Diese sind dann von weiteren Komponenten zu konsumieren. Diese praktische Weiterverarbeitung der gewonnenen Daten wurde jedoch nicht näher beleuchtet.

Nach der Entwicklung des Prototypen wurde aber in Zusammenarbeit mit der italienischen IT Dienstleistungsfirma SighUp dieser evaluiert. Es konnte festgestellt werden, dass abgeleitete High-Level Events, durch zuvor definierte Szenarien erstellt werden und Informationen beinhalten, die für die Problemfindung in diesen Szenarien hilfreich ist.

Der entwickelte Prototyp ist quelloffen unter <https://github.com/marioscrock/Kaiju> verfügbar. Dieser wurde aber seit Juli 2020, dem Zeitpunkt der Veröffentlichung des Papers, nicht mehr aktualisiert. Weiterhin lässt sich auch kein Folgeprojekt finden. Auch wenn Kaiju kein praktisches Werkzeug für die Lösung in dieser Arbeit darstellt, so wird ein gutes Bild über die technische Seite der Nachvollziehbarkeit vermittelt, sowie wird zuvor der Forschungsstand erläutert.

3.3.2.6 Fazit zur Literatur

Wie in den verschiedenen beleuchteten Berichten und Arbeiten zu lesen ist, von denen drei in 2020 veröffentlicht wurden, ist das Forschungsfeld der Nachvollziehbarkeit bzw. der Observability bisher nur spärlich untersucht worden. Eigene Recherchen ergaben das gleiche Ergebnis, das Feld enthält wenige konkrete Veröffentlichungen. Speziell zur Nachvollziehbarkeit im Bereich der Frontend-Entwicklung konnte keine Arbeit gefunden werden.

Es konnte jedoch ein Konsens identifiziert werden, dass dieses Forschungsfeld einen wichtigen Mehrwert darstellt, speziell bei komplexen Microservice-Architekturen. Weiterhin lässt sich ein steigender Trend in konkreten Lösungsansätzen im Forschungsfeld identifizieren, wie u. A. die in dieser Arbeit näher betrachteten Veröffentlichungen. Diese Entwicklung geht einher mit den Standardisierungsbemühungen aus der Wirtschaft rund um OpenTelemetry.

4 Erstellung Proof-of-Concept

4.1 Anforderungen

4.1.1 Definitionen

Um die Anforderungen systematisch zu einzuordnen, werden folgend zwei Modelle vorgestellt, anhand dessen die Kategorisierung erfolgt.

Beim ersten Modell handelt es sich um das Kano-Modell [Kan68] der Kundenzufriedenheit kategorisiert (vgl. Tabelle 4.1).

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

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

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

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

Tab. 4.2: Kategorien der Anforderungen

4.1.2 Anforderungsanalyse

Die erste Quelle von Anforderungen ergibt sich durch die wissenschaftliche Fragestellung und somit durch den Autor selbst. Jedoch stellt die primäre Quelle von Anforderungen die Stakeholder selbst dar. Die Stakeholder dieser Arbeit sind Christian Wansart und Stephan Müller von Open Knowledge. Sie betreuen die Arbeit und haben ein starkes Interesse, dass ein sinnvolles und übertragbares Ergebnis aus der Arbeit entspringt, um es z.B. bei Kunden anwenden zu können. Mit den beiden Stakeholdern wurde sich mindestens wöchentlich zu einer (virtuellen) Diskussionsrunde getroffen, um die bisherigen Ergebnisse zusammenzutragen, das weitere Vorgehen zu besprechen und auch um Lösungsansätze zu diskutieren. In der Motivation wurde ein Kunde der Open Knowledge beschrieben, welcher von den Ergebnissen dieser Arbeit profitieren könnte. Die beiden Stakeholder arbeiten u. A. auch für diesen Kunden und brachten Wünsche und betriebliche Gegebenheiten des Kunden in die wöchentlichen Diskussionsrunden mit ein. Diese indirekte Kommunikation stellt die zweite Quelle an Anforderungen dar.

Anforderungen können neben einer auch eine Kombination von mehreren Quellen besitzen, wenn die Anforderung aus einer gemeinsamen Bestrebung oder Diskussion entstand. In den Anforderungen werden die Quellen mit Kürzeln angegeben, vgl. Tabelle 4.3.

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

Tab. 4.3: Quellen der Anforderungen

4.1.3 Anforderungsliste

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

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

Tab. 4.4: Beispiel einer Anforderung

4.1.3.1 Grundanforderungen

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

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

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

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

4.1.3.2 Funktionsumfang

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

Id	Name	Kano-Modell	Funktionsart	Quelle
2011	Use-Case-Logging	B	f	S
Tritt ein Use-Case auf, soll dieser im Log notiert werden. Beispielsweise soll notiert werden, dass ein Nutzer einen neuen Datensatz angelegt möchte und wenn der diesen anlegt.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2020	Error-Monitoring	B	f	S
Tritt ein Fehler auf, der nicht gefangen wurde, so ist dieser automatisch zu erfasst und um weitere Attribute zu ergänzen. Sonstige Fehler können auch erfasst werden, aber hierbei Bedarf es einem manuellen Aufruf einer Schnittstelle				

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

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

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

4 Erstellung Proof-of-Concept

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

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

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

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

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

Id	Name	Kano-Modell	Funktionsart	Quelle
2160	Datadump	S	f	S
Möglichkeit zum Export des fachlichen Modells des Frontends.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2161	Datadump-Import	S	f	S
Re-Import des fachlichen Modells des Frontends, um diesen Zustand auf anderen Systemen und für andere Systeme einsehbar zu machen.				

4.1.3.3 Eigenschaften

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

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

Id	Name	Kano-Modell	Funktionsart	Quelle
3100	Anzahl Partnersysteme	B	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	L	f	A+S
Das Logging von Werten soll auf einem vordefinierten Format. Für ähnliche Funktionsgruppen (wie ein Schnittstellenaufwurf) soll das gleiche Format verwendet werden. Ein anwendungsübergreifendes Format ist nicht gefordert.				

4.1.3.4 Partnersysteme

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

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

4 Erstellung Proof-of-Concept

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

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

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

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

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

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

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

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

4.2 Vorstellung der Demoanwendung

Wie in Anforderung 1020 beschrieben, ist eine Demoanwendung zu erstellen, auf Basis dessen das Konzept anzuwenden ist und somit praktisch umgesetzt werden kann. Dieser Abschnitt beschäftigt sich mit der Vorstellung der Demoanwendung und der repräsentativen Aufgabe, die diese übernimmt.

In der Motivation wurde ein konkretes Problem eines Kunden der Open Knowledge genannt. Damit die Demoanwendung realistisch eine mdoerne Webanwendung darstellt, wird sie in Grundzügen die Webanwendung des Direktversicherers nachahmen. Bei der Webanwendung handelt es sich um eine mit Angular erstellte SPA, die den Nutzer verschiedene teils dynamische Formulare ausfüllen lässt und am Ende diese Daten an einen Dienst sendet und ein Ergebnis erhält, welches dargestellt wird. Während der Eingabe der Formulare werden einzelne Werte gegen Dienste validiert.

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

4.2.1 Verhaltensdefinition

Mit den beiden Stakeholdern, also Christian Wansart und Stephan Müller, die beide am Projekt für den Kunden involviert sind, wurde diese Verhaltensdefinition erstellt. Diesen Ansatz der Definition der Software anhand des Verhaltens nennt man Behaviour Driven Development (BDD). Um die BDD-Definition festzuhalten wurde sie in der gängigen

Gherkin [Sma19] Syntax geschrieben. Die Syntax ist natürlich zu lesen, folgend werden alle gewünschten Features der Demoanwendung in der Gherkin-Syntax beschrieben.

```
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 prüft
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
29            * die Stadt eingeben zu können
30            * 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 der Hausnummer eingegebenen
33 When sie auf "weiter" klickt
34 Then soll sie informiert werden, dass sie alle Felder ausfüllen
35 muss
36
37 Scenario: Kundin gibt invalide Daten ein
38 When die Kundin eine andere Rechnungsadresse eingibt
39 * Vorname und Nachname Sonderzeichen enthalten außer Bindestriche
40 enthält
41 * Straße Sonderzeichen außer Bindestriche und Punkte enthält
42 * Hausnummer. Sonderzeichen enthält
43 * PLZ alles andere außer Zahlen enthält
44 * Stadt keine deutsche Stadt ist
45 * das @ bei der E-Mail-Adresse fehlt
46 Then soll eine Warnung angezeigt werden und der "weiter"-Button
47 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
22 Then sollen die zuvor eingegebenen Adressdaten weiterhin vorhanden
23 sein
24
```

```
15 Scenario: Kundin möchte die Rechnungsadresse übernehmen
16 Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"
    ist gesetzt
17 When sie auf den "weiter"-Button klickt
18 Then soll sie auf die Seite "Zahlungsdaten" gelangen
19
20 Scenario: Kundin möchte andere Lieferdaten nutzen
21 Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"
    wurde entfernt
22 When die Kundin hat alle Felder ausgefüllt
23 And sie auf den "weiter"-Button klickt
24 Then soll sie auf die Seite "Zahlungsdaten" gelangen
25
26 Scenario: Kundin möchte andere Lieferdaten nutzen, ohne alle Felder
    ausgefüllt zu haben
27 Given das Häkchen bei "Gleiche Lieferdaten wie Rechnungsadresse"
    wurde entfernt
28 When die Kundin eine andere Lieferdaten eingibt
29 * Vorname und Nachname Sonderzeichen enthalten außer Bindestriche
    enthält
30 * Straße Sonderzeichen außer Bindestriche und Punkte enthält
31 * Hausnummer. Sonderzeichen enthält
32 * PLZ alles andere außer Zahlen enthält
33 * Stadt keine deutsche Stadt ist
34 * das @ bei der E-Mail-Adresse fehlt
35 And sie auf den "weiter"-Button klickt
36 Then soll eine Warnung angezeigt und der "weiter"-Button blockiert
    werden
```

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

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

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

```
1 Feature: Bestellung abschließen
```

```
2 |
3 |   Die letzte Seite soll eine Übersicht über die zuvor eingegebenen Daten
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 des eigentlichen User-Interfaces soll auch ein Backend teil der Demoanwendung sein. Hierfür wurde auf Basis der Verhaltensdefinition eine Architektur entworfen, die im folgenden Abschnitt näher beschrieben wird.

4.2.2 Backend

Das Backend wurde als Microservice-Architektur konzipiert und wurde ebenso wie die Webanwendung auch an das Projekt des Open Knowledge Kunden angelehnt. In Abbildung 4.1 lässt sich die konzipierte und umgesetzte Architektur betrachten, dargestellt als Kubernetes [The20] Deployment Diagramm.

Durch diese recht komplexe Architektur kann eine realitätsnahe Repräsentation erfolgen. Speziell wird bei einer solchen Architektur Tracing hilfreicher, um die Zusammenhänge zwischen den Diensten nachvollziehen zu können. Dies wird beim Einsatz und der Vorstellung der Lösung näher betrachtet.

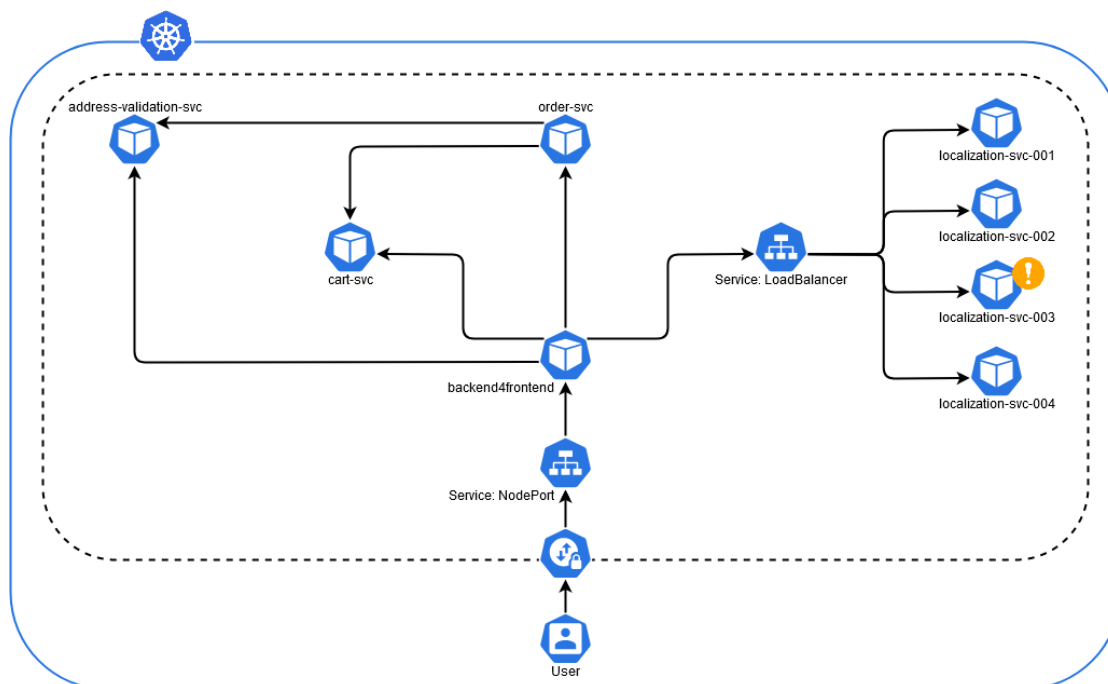


Abb. 4.1: Demoanwendung: Kubernetes-Architektur-Diagramm, Quelle: Eigene Darstellung

4.2.3 Frontend

Das Frontend stellt das Herzstück der Demoanwendung dar und wurde auf Basis der Verhaltensdefinition erstellt. Wie beim Direktversicherer wurde es mit Angular erstellt und als zustandsreiche SPA umgesetzt. Genauere Implementierungsdetails werden jedoch nicht behandelt, außer sie sind relevant für die Lösung oder die Lösungsüberprüfung.

Das Frontend sowie das Backend wurden mit einigen Fehlern implementiert. Hintergrund ist, dass bei der Betrachtung der Lösung solche Fehler notwendig sind, um diese zu bewerten. Diese Fehler werden näher im Unterabschnitt 4.2.4 betrachtet.

Die einzelnen Features der Verhaltensdefinition wurden als Seiten umgesetzt und das Layout verknüpft diese Seiten in einem blätternen Format, bei dem man vor- und zurückspringen kann. Dies kann in den folgenden 6 Abbildungen betrachtet werden.

4 Erstellung Proof-of-Concept

1 Warenkorb




2 Rechnungsadresse

3 Lieferdaten

4 Zahlungsdaten

5 Bestellung abschließen

Warenkorb

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

Bestellvorgang starten

2021, Mustermann GmbH & Co. KG (fiktiv), 44141 Dortmund

Bilder sind unter der [Pixabay Lizenz](#) lizenziert.

Abb. 4.2: Demoanwendung: Startseite „Warenkorb“

1 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

Zurück

Weiter

2021, Mustermann GmbH & Co. KG (fiktiv), 44141 Dortmund

Bilder sind unter der [Pixabay Lizenz](#) lizenziert.

Abb. 4.3: Demoanwendung: Seite „Rechnungsadresse“

4 Erstellung Proof-of-Concept

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

Lieferdaten

☐ Gleiche Lieferadresse wie Rechnungsadresse

Anrede
Herr

Vorname
Peter

Nachname
Mustermann

Straße
Musterring

Nr.
1337

Postleitzahl
44135

Stadt
Dortmund

5 / 5

Zurück **Weiter**

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

Abb. 4.4: Demoanwendung: Seite „Lieferdaten“

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“

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

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

Abb. 4.6: Demoanwendung: Seite „Bestellübersicht“

Bestellbestätigung

Vielen Dank für Ihre Bestellung!
Ihre Bestellung von 3 Artikeln wurde aufgenommen und wird unter der Nummer #6509 bearbeitet.
In Kürze werden wir 9.30 € über Ihre gewählte Rechnungsart "Rechnung" einholen.
Die Artikel werden Ihnen in voraussichtlich 3-5 Werktagen zugestellt, unter Berücksichtigung folgender Adresse:

Herr Peter Mustermann
Musterring 1337
44135 Dortmund

Zum Shop

Abb. 4.7: Demoanwendung: Finale Seite „Bestellbestätigung“

4.2.4 Fehlerszenarien

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

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

4.2.4.1 „Keine Übersetzungen“

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

- Ursache: Einer der „localization-svc“ Pods hat eine defekte Konfiguration, dieser ist in Abbildung 4.1 visuell hervorgehoben. Wird man durch den LoadBalancer mit diesem Pod verbunden tritt das Fehlverhalten auf. Dies ist eine Nachstellung eines tatsächlichen Problems beim Kunden.



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

Abb. 4.8: Fehlende Texte

4.2.4.2 „Gültige Straßen sind ungültig“

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

- Ursache: Der „address-validation-svc“ validiert Straßen mit dem RegEx `[a-zA-Z\,\-\-]+`, welches keine gängigen Sonderzeichen (ä ,ö ,ü, ß) erlaubt.

4.2.4.3 „Gültige Hausnummern sind ungültig“

- Problem: Nutzer berichten, dass Hausnummern, die nicht nur aus Zahlen bestehen, zum Fehler führen.

- Ursache: Der „address-validation-svc“ validiert Hausnummern als Zahl und schlägt im o. g. Fall in der Konvertierung fehl.

4.2.4.4 „Gültige Städte sind ungültig“

- Problem: Nutzer aus Gießen berichten, dass Sie das Formular zur Rechnungsadresse nicht ausfüllen können
- Ursache: Der „address-validation-svc“ meldet die Stadt „Gießen“ als ungültig, weil sie nicht in der lokalen Tabelle vorhanden ist.

4.2.4.5 „Ungültige Adressen sind gültig“

- Problem: Nutzer können in den Lieferdaten ungültige Eingaben tätigen und absenden, bei der Bestellaufgabe kommt es zu einem Fehler.
- Ursache: Das Frontend überprüft lediglich die Rechnungsadresse, aber nicht die Lieferadresse

4.2.4.6 „Vor- und Nachnamen werden abgeschnitten“

- Problem: Nutzer berichten, dass in der Bestellbestätigung Ihre Vor- und Nachnamen abgeschnitten dargestellt werden.
- Ursache: Der „order-svc“ begrenzt den Vor- sowie den Nachnamen auf 20 Zeichen.

4.2.4.7 „Falsche Zahlungsart“

- Problem: Nutzer berichten, dass in der Bestellbestätigung die falsche Zahlungsart angezeigt wird. In der Bestellübersicht wurde jedoch die korrekte Zahlungsart angezeigt.
- Ursache: Das Frontend sendet alle Formulardaten an „order-svc“, dieser aber an, dass alle nicht ausgewählten Formulare `null` sind.

4.2.4.8 „Lange Verarbeitung“

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

4.2.5 Repräsentation

Eine wichtige Eigenschaft der Demoanwendung sollte sein, dass sie repräsentativ für das Kundenprojekt und im Allgemeineren auch für SPAs ist. Durch den groben Aufbau der Demoanwendung, also einer zustandsreichen und größtenteils clientbasierten SPA, stellt die Demoanwendung eine moderne Webanwendung dar. Auch die Verwendung von Angular ist repräsentativ, denn Angular ist eines der meist verwendeten Frontend-Frameworks [GB20].

TODO: Expand this

4.3 Konzept

4.3.1 Datenverarbeitung

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

4.3.1.1 Erhebung

Im Standardfall erhalten Betreiber und Entwickler, bis auf die Kommunikation mit Partnersystemen, keine Informationen von einer SPA. Deswegen sollen zunächst Loginformationen des Frontends erhoben und an ein weiteres System gesendet werden. Hierbei ist eine Unterscheidung sinnvoll, welche Logmeldungen tatsächlich gesendet werden sollen (bspw. anhand der Log Kritikalität). Diese Unterscheidung sollte konfigurativ änderbar sein. Dieser Datenstrom wird erfahrungsgemäß unregelmäßig aber doch schon sehr häufig mit Daten befüllt, um eine gute Nachvollziehbarkeit zu erreichen.

Neben den Loginformationen sollten nicht gefangene Fehler und optional gefangene Fehler an ein weiteres System weitergeleitet werden. Dabei sollen alle relevanten und zugreifbaren Informationen mitgesendet werden.

Eine Datenerhebung wie beim Real-User-Monitoring, wo jede Benutzerinteraktion aufgezeichnet wird, um bspw. Klickpfade zu optimieren oder um festzustellen wie lange ein Nutzer sich auf einer Seite aufhielt. Jedoch ist ein Session-Replay Mechanismus enorm hilfreich und gewünscht, welcher ebenso jede Benutzerinteraktion aufzeichnen muss. Damit nicht zu viele Daten erhoben werden und damit nicht jede Nutzersitzung mitgeschnitten wird, soll die Aufzeichnung erst nach Einwilligung und Aktivierung seitens des Nutzers erfolgen oder bei speziellen Umgebungen automatisch (bspw. einer Staging-Umgebung). Weiterhin könnten die Loginformationen nach dieser Einwilligung auch feingranularer übertragen werden, bspw. ohne Einwilligung würden Logs der Kritikalität INFO und höher übertragen werden und mit Einwilligung auch Logs der Kritikalität DEBUG und höher.

Des Weiteren soll ein Tracing der Kommunikation zwischen Frontend und Partnersystemen eingesetzt werden. Hierbei könnten auch wichtige Verarbeitungsmethoden im Tracing erfasst werden, dies wird jedoch hier nicht definiert und ist Teil der eigentlichen Implementierung. Es soll, wenn möglich, auf den neuen Standard OpenTelemetry (OTel) aufgesetzt werden. Hierdurch wird das Erheben von Tracing- und Metrikdaten standardisiert und zukünftig auch für Logdaten möglich. Auf Basis von OTel sollen beispielhaft

Metriken erfasst werden, um das Anwendungsverhalten nachzuhalten und zu überwachen. Durch diese Metriken können Aspekte eines Application Performance Monitorings erfüllt werden.

Alle gesendeten Datensätze sollen möglichst mit Metadaten angereichert werden, die einerseits den Nutzer, die Umgebung und die Anwendung identifizieren. Diese umfassen u. A.: Zeitstempel, Session-Id, User-Agent, IP, Hostsystem, Version.

4.3.1.2 Auswertung

Bei der Auswertung der Daten soll hauptsächlich auf bestehende Technologien aufgebaut werden, wie z.B. die Technologien, die in Abschnitt 3.3 evaluiert wurden. Das heißt im Konzept kann nicht im Detail darauf eingegangen werden, wie diese Daten tatsächlich verarbeitet werden und dies ist auch nicht direkt von Relevanz. Eine Beschreibung folgt im Unterabschnitt 4.3.3 sowie beim Einsatz von den Technologien selbst.

Lediglich bei der Vorverarbeitung des Tracings im Client kann nun bereits eine Aussage getroffen werden. Wird hierbei nämlich, wie gewünscht, auf OTel gesetzt, dann erfolgt eine Vorverarbeitung von den Komponenten von OTel selbst. Dabei werden u. A. die Spankontexte fürs Tracing und die Beziehung zwischen den Spans gepflegt.

4.3.1.3 Visualisierung

Wie bei der Auswertung wird auch die Visualisierung stark abhängig von den eingesetzten Technologien sein. Nichtsdestotrotz können bereits hier gewünschte Ansichten/Funktionen definiert werden:

- Die Logdaten sollen abrufbar sein und filterbar sein.
- Fehlerinformationen sollen gesondert der Logdaten dargestellt werden.
 - Fehler sollen gruppiert werden, um gleiche Fehlerbilder zusammenzufassen.
 - Zu den Fehlerbildern sollen übergreifende Informationen dargestellt werden.
 - Es lassen sich auch einzelne Fehler einer Fehlergruppe anzeigen.
- Für einen ausgewählten Trace soll ein Trace-Gantt-Diagramm dargestellt werden (vgl. Unterabschnitt 3.1.3)
- Die beispielhaften Metriken soll visuell dargestellt werden.
- Die Daten zum Session-Replay sollen, wenn vorhanden, visuell dargestellt werden, sodass die Interaktionen des Nutzers nachzuvollziehen sind.

4.3.2 Architektur

Auf Basis der zuvor definierten Grundkonzepte zur Datenverarbeitung, wird nun eine grobe Architektur konzipiert. Im Allgemeinen sollen die Funktionsbereiche Log Management, Error Monitoring, Application Monitoring, Tracing sowie Session-Replay erfüllt werden. Im Client soll es für jeden Funktionsbereich eine eigene Komponente geben, die die jeweiligen Daten erfasst und an das entsprechende Partnersystem weiterleitet. Lediglich die Erfassung von Metriken und Tracing soll auf Basis von OpenTelemetry erfolgen und daher dieselben Komponenten verwenden. Dieser Aufbau lässt sich auf der linken Seite der Abbildung 4.9 betrachten.

Es wurde nach Anforderung 3100 versucht die Anzahl der Partnersysteme gering zu halten. Für die Datenverarbeitung, -analyse und -visualisierung von Logs, Fehlern und Metriken soll ein einzelnes System zuständig sein. Denn für die Bewältigung dieser verschiedenen Kategorien sind ähnliche Disziplinen notwendig, wodurch ein einzelnes System ausreichen sollte. Grundlegende Funktionen dieses Systems belaufen sich auf die Langzeitspeicherung, eine performante Suche und die Visualisierung in Graphen.

Für Tracing wird ein zweites System benötigt, hauptsächlich weil in der Evaluation kein Werkzeug identifiziert werden konnte, welches neben den zuvor genannten 3 Datenkategorien auch Tracingdaten gut unterstützt. Tracingdaten sind zudem anders, dadurch dass sie einen hohen Datendurchsatz und -menge aufweisen.

Ein drittes System soll die Funktionalität rund um Session-Replay übernehmen. Hierbei liegt auch der Grund darin, dass kein Werkzeug identifiziert werden konnte, welches neben Session-Replay auch andere Disziplinen erfüllen kann. Weiterhin sind, wie beim Tracing, die Eigenschaften der Daten anders, denn hier werden nahezu konstant Daten versendet, um alle Benutzerinteraktionen und das Anwendungsverhalten nachstellen zu können.

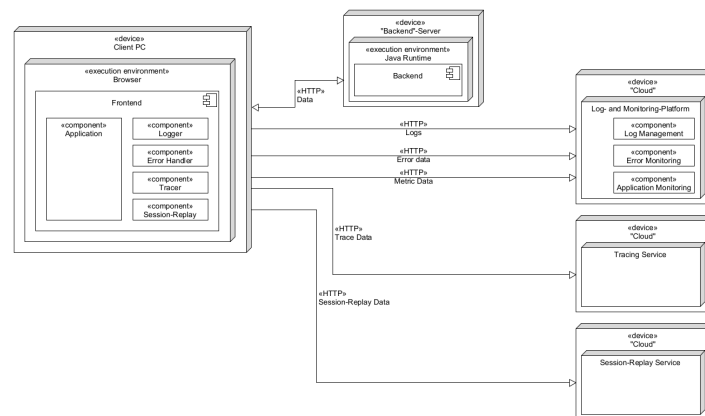


Abb. 4.9: Grobe Architektur

Wie in der Datenerfassung erwähnt, werden die einzelnen Datentypen unterschiedlich erhoben und besitzen somit auch andere Eigenschaften. Wie IBM bei Big Data definiert [ZE11], lassen sich auch hier die Eigenschaften Volume, Velocity und Variety identifizieren. Der Aspekt Volume ist weniger präsent, denn die Datenmengen sind nicht vergleichbar mit echten Big-Data-Anwendungen. Genau ist dies nicht prognostizierbar, aber in der Evaluierung des Stands der Technik, ließ sich ein Datendurchsatz von 1 MB/-min feststellen - somit stellt dies im Frontend keine Herausforderung dar, jedoch in den verarbeitenden Systemen kann dies natürlich durch eine große Menge an Frontends multipliziert werden, was jedoch nicht im Fokus der Arbeit steht.

Eine Variety der Daten, also Unterschiedlichkeit der Datenstruktur, ist definitiv vorhanden und entspringt den verschiedenen Funktionsgebieten. Auch innerhalb derselben Datenströme kann eine Variety identifiziert werden, denn bspw. sind Logmeldungen sehr individuell, sie folgen meist nicht streng einem Format und enthalten unterschiedliche Mengen an Informationen.

Der Aspekt des Velocity ist zudem auch sehr wichtig und eine Visualisierung dessen für das vorhergehende Konzept findet sich in Abbildung 4.10.

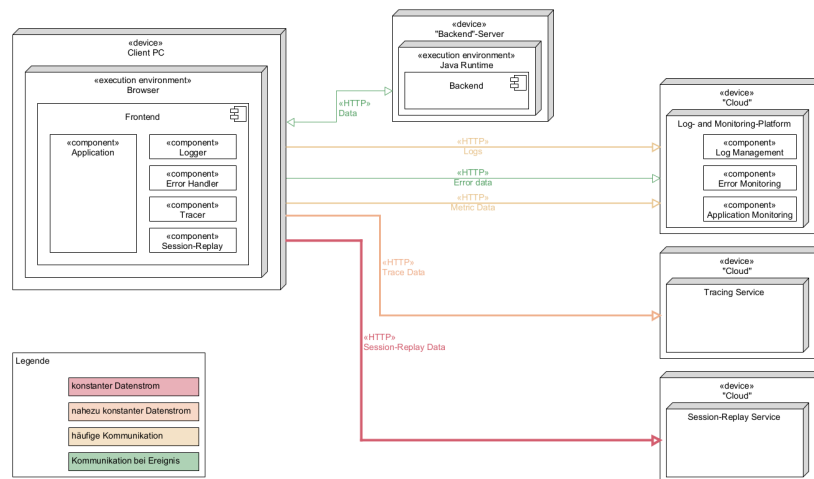


Abb. 4.10: Grobe Architektur mit hervorgehobenem Datendurchsatz

4.3.3 Technologie-Stack

Im Frontend sind für den Logger und den Error-Handler keine speziellen Technologien zu verwenden, es soll somit auf die bestehende Demoanwendung aufgesetzt werden. Da die Demoanwendung in Angular geschrieben ist, soll ein ErrorHandler implementiert werden, welcher die Information dann weiterleitet. Die Tracing- und Metrikdaten werden mit OpenTelemetry JavaScript-Komponenten erfasst und an ein Partnersystem gesendet. Für die Session-Replay-Daten wird jedoch auf eine proprietäre Komponente gesetzt, denn bei der Evaluierung konnte keine quelloffene Datenerhebung identifiziert werden.

Als Log- und Monitoringsystem soll Splunk zum Einsatz kommen. Nach der Evaluation von Splunk lassen sich hiermit gut die drei Datenkategorien Logs, Metriken und Fehler speichern, analysieren und visualisieren. Alternativen wären u. A. Dynatrace und New Relic, basieren jedoch auf einer proprietären Datenerhebung. Diese bieten eher ausgefertigte Dashboards und Graphen, mit zu weniger Flexibilität zur Datenvisualisierung. Honeycomb wäre zudem auch eine Alternative, eignet sich aber weniger für strikte Logdaten. Unter anderem werden die Daten bei Honeycomb nur für 60 Tage gespeichert. Somit blieb Splunk als einziger Kandidat, der den gewünschten Rahmen erfüllt. Es sollten jedoch auch andere Werkzeuge Splunk ersetzen können, bspw. der Elastic Stack könnte hierfür auch geeignet sein, dieser wurde jedoch nicht zuvor evaluiert.

Für das Partnersystem, welches Session-Replay übernimmt, wurde nur ein Werkzeug evaluiert, nämlich LogRocket. Da diese Art von Software als sehr zielführend empfunden wurde, wird LogRocket somit in der Lösung verwendet. Jedoch ist anzumerken, dass ähnlich wie bei Splunk, LogRocket nicht zwingend vorgeschrieben ist, sondern gegen ein äquivalentes Werkzeug austauschbar ist (vgl. Unterunterabschnitt 3.3.1.9). LogRocket kann jedoch auch überzeugen, denn es bietet Funktionen, um datenschutzkritische Aspekte zu beachten aber auch um die Performanz der eigentlichen Anwendung nicht zu stark einzuschränken. So wird LogRockets eigentliche Logik erst nach dem Seitenladen dynamisch nachgeladen und die Hauptarbeit findet generell in Worker-Threads statt, nicht im Hauptthread [Log20b]. Des Weiteren schränkt LogRocket die Aufnahmequalität ein, wenn eine unzureichende Bandbreite bemerkt wird.

Allgemein werden bei LogRocket zudem Daten nicht im JSON-Format kommuniziert, sondern mit Google Protobufs (protocol buffers) [Goo20b], welche effizienter sind (siehe Abbildung 4.11). Die Verwendung von Protobufs könnte ggf. auch interessant sein bei der Datenübertragung der anderen Datenkategorien, jedoch wird sich an dieser Stelle nicht darauf festgelegt. Sollte eine hohe Übertragungseffizienz tatsächlich vonnöten sein, so werden Protobufs ggf. näher betrachtet.

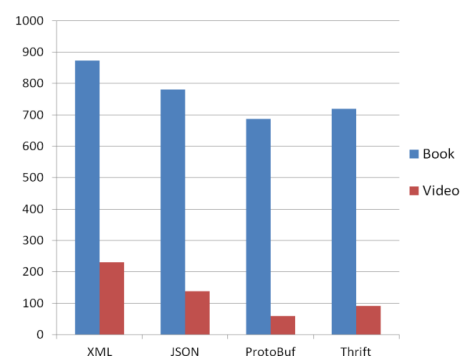


Abb. 4.11: Serialisierte Datengröße je Format [SM12]

Um die Tracingdaten zu beleuchten und so Einblick in den architektonischen Verlauf eines Aufrufs zu erhalten, wird Jaeger verwendet. Jaeger bietet jene Tracing-Visualisierungen, die Splunk fehlen und ist zudem darauf spezialisiert Tracingdaten zu speichern und zu analysieren. Um diese Funktionen anzubieten und eine schnelle Datenabfrage zu gewährleisten, besitzt Jaeger eine darauf konzipierte Infrastruktur (vgl. Abbildung 4.12). Die verwendete Datenbank ist Cassandra und für Analyse- und Verarbeitungszwecke wird Apache Spark benutzt.

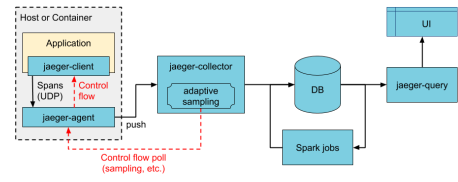


Abb. 4.12: Architektur von Jaeger, angepasst [Jae20a]

Die geplante Architektur ist in Abbildung 4.13 zu betrachten.

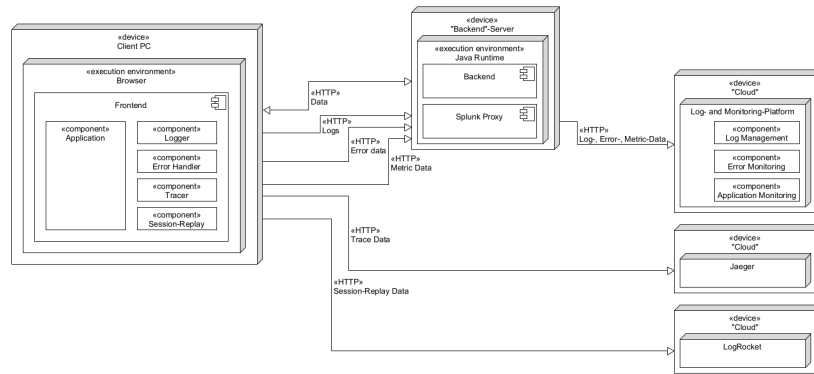


Abb. 4.13: Architektur mit speziellen Technologien

4.3.4 Übertragbarkeit

Übertragbarkeit beschäftigt sich mit der Eigenschaft eines Ansatzes oder Konzeptes auch für andere Softwareprojekte anwendbar zu sein. Grundlegend muss genannt werden, dass das Konzept sich auf Softwareprojekte, die JavaScript-basierte Webanwendungen enthalten, beschränkt. Somit wird auch die Übertragbarkeit auch nur für ähnliche Softwareprojekte bewertet.

Da das Konzept nicht eine Basisstruktur darstellt, auf das man ein Projekt basieren kann, sondern viel eher ein bestehendes Projekt erweitert, lässt sich das Konzept bei realen Projekten in verschiedensten Phasen integrieren. Weiterhin wurden die Grundkonzepte, wie die Erhebung von Tracingdaten, nicht auf etablierte Technologien basiert, sondern viel eher auf die zugrundeliegenden Disziplinen. Deswegen sind alle vorgeschlagenen Technologien und Partnersysteme austauschbar, um verschiedene Projekt- und

Unternehmensvorgaben zu erfüllen. Hinzukommend ist das Konzept modular zu verstehen, denn einzelne Disziplinen werden von spezialisierten Systemen gehandelt und sind somit auch entfernbar. Diese Flexibilität der Auswahl an Komponenten und der Zusammensetzung verspricht Projektteams eine reell umsetzbare Lösung.

Ein identifizierbares Manko ist jedoch die Menge an Partnersystemen, die die Lösung vorschlägt. Dies kann aus Wartungs- oder Kostengründen nicht für jedes Projektteam attraktiv sein, jedoch konnte leider keine weitere Verringerung der Partnersysteme erzielt werden. Wenn man aber auf einer der Datenkategorien verzichten kann, dann ließe sich ggf. auch ein Partnersystem entfernen. Sollte aber eine Technologie existieren oder sich über Zeit entwickeln, welche mehrere Disziplinen löst, so kann die Anzahl ebenso verringert werden. Dabei anzumerken ist aber auch, dass damit eine gewisse Flexibilität verloren geht. Es lässt sich aber nicht definitiv eine Bewertung hierzu erstellen, da eben solch eine Technologie nicht gefunden werden konnte.

Alles in Allem, sollte das vorgeschlagene Konzept aber für den Großteil von Softwareprojekten in der Frontendentwicklung adaptierbar sein. Eine tiefergehende Betrachtung der Übertragbarkeit erfolgt in Abschnitt 5.3, auf Basis der tatsächlichen Implementierung. Diese Übertragbarkeit wird wahrscheinlich nicht gleich bleiben, denn die Implementierung wird bspw. einige spezifische Entscheidungen beinhalten, die eine Übertragbarkeit behindern könnten.

4.4 Implementierung

Auf Basis des Konzeptes soll nun eine Implementierung erfolgen.

5 Ergebnis

5.1 Demonstration

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

5.2 Kriterien

Die zuvor definierten Kriterien in 4.1 sollen hier überprüft werden.

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

5.3 Übertragbarkeit

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

5.4 Einschätzung von anderen Entwicklern (optional)

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

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

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

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

6 Abschluss

6.1 Fazit

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

6.2 Ausblick

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

7 Anhang

7.1 Studien zur Browserkompatibilität

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

Für die Suche wurde folgender Suchterm benutzt:

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

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

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

Tab. 7.1: Suchtreffer zu Studien über Browserkompatibilität

Eidesstattliche Erklärung

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

Dortmund, am
(Unterschrift)

Abkürzungs- und Erklärungsverzeichnis

Ajax Asynchronous JavaScript and XML

CDN Content Delivery Network

Clientseitiges Rendering Der Server stellt dem Client lediglich die Logik und die notwendigen Daten bereit, die eigentliche Inhaltsgenerierung geschieht im Client. Für ein Beispiel siehe Unterabschnitt 2.2.2

CNCF Cloud Native Computing Foundation

CORS Cross-Origin Resource Sharing

CPU Central Processing Unit, auf Deutsch „Prozessor“.

CSP Content-Security-Policy

ESP Event-Stream-Processor

HTTP Hyper-Text-Transfer-Protocol

OTel OpenTelemetry

PoC Proof-of-Concept

SaaS Software-as-a-Service

SDLC Software-Development-Life-Cycle

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

W3C World Wide Web Consortium

XHR XMLHttpRequest

XSS Cross-Site-Scripting

Abbildungsverzeichnis

2.1	Studien zur Browserkompatibilität, eigene Darstellung (vgl. 7.1)	4
2.2	Flowchart über den Ablauf von Ajax-Anfragen mit CORS [Blu15]	8
3.1	Kausale Beziehung zwischen Spans. Eigene Darstellung.	11
3.2	Zeitliche Beziehung zwischen Spans. Eigene Darstellung.	11
3.3	Fehlerbericht in der Instagram App [Fac20a]	12
3.4	Schaubild einer Lösung auf Basis von OTel [Ope20c]	15
3.5	OTel Komponenten [Dyn20c]	15
3.6	Beispiel eines Session-Replays bei LogRocket	18
3.7	Trace-Detailansicht. Quelle: Don Schenck [Sch20]	20
3.8	Dienst-Abhängigkeits-Graph. Quelle: Yuri Shkuro [Shk20]	20
3.9	Zoom von Abbildung 3.10, Abbildung aus [ASD ⁺ 20]	21
3.10	TraVistas Gantt-Diagramm, Abbildung aus [ASD ⁺ 20]	22
3.11	Mitschneiden von DOM-Events, Abb. aus [BBKE13]	23
3.12	Abspielen von DOM-Events, Abb. aus [BBKE13]	23
3.13	Übersicht von FAME, Abb. aus [OS ⁺ 18]	24
3.14	Kaiju Architektur, Abb. aus [STM ⁺ 20b]	25
4.1	Demoanwendung: Kubernetes-Architektur-Diagramm, Quelle: Eigene Darstellung	39
4.2	Demoanwendung: Startseite „Warenkorb“	40
4.3	Demoanwendung: Seite „Rechnungsadresse“	40
4.4	Demoanwendung: Seite „Lieferdaten“	41
4.5	Demoanwendung: Seite „Zahlungsdaten“	41
4.6	Demoanwendung: Seite „Bestellübersicht“	42
4.7	Demoanwendung: Finale Seite „Bestellbestätigung“	42
4.8	Fehlende Texte	43
4.9	Grobe Architektur	48
4.10	Grobe Architektur mit hervorgehobenem Datendurchsatz	49
4.11	Serialisierte Datengröße je Format [SM12]	50
4.12	Architektur von Jaeger, angepasst [Jae20a]	51
4.13	Architektur mit speziellen Technologien	51

Tabellenverzeichnis

4.1	Merkmale nach dem Kano-Modell der Kundenzufriedenheit	27
4.2	Kategorien der Anforderungen	28
4.3	Quellen der Anforderungen	28
4.4	Beispiel einer Anforderung	29
7.1	Suchtreffer zu Studien über Browserkompatibilität	56

Quellcodeverzeichnis

4.1	Demoanwendung: Gherkin Definition zum Feature „Warenkorb“	35
4.2	Demoanwendung: Gherkin Definition zum Feature „Rechnungsadresse“ . . .	35
4.3	Demoanwendung: Gherkin Definition zum Feature „Lieferadresse“	36
4.4	Demoanwendung: Gherkin Definition zum Feature „Zahlungsdaten“	37
4.5	Demoanwendung: Gherkin Definition zum Feature „Bestellung abschließen“	37

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
- [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
- [AS20] ANAND, Vaastav ; STOLET, Matheus: TraViz: Visualization of Traces in Distributed Systems / University of British Columbia. 2020. – Forschungsbericht
- [ASD⁺20] ANAND, Vaastav ; STOLET, Matheus ; DAVIDSON, Thomas ; BESCHASTNIKH, Ivan ; MUNZNER, Tamara ; MACE, Jonathan: Aggregate-Driven Trace Visualizations for Performance Debugging. In: *arXiv preprint arXiv:2010.13681* (2020)
- [AW20] ANAND, Vaastav ; WONSIL, Joseph: Tracey - Distributed Trace Comparison and Aggregation using NLP techniques / University of British Columbia. 2020. – Forschungsbericht
- [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 09.11.2020]
- [Bra16] BRAUN, Michael: Nicht-funktionale Anforderungen. In: *Juristisches IT-Projektmanagement Lehrstuhl für Programmierung und Softwaretechnik Ludwig-Maximilians-Universität München* (2016). – S. 3-5
- [Dyn20a] DYNATRACE: *Dynatrace joins the OpenTelemetry project*. <https://www.dynatrace.com/news/blog/dynatrace-joins-the-opentelemetry-project/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Dyn20b] DYNATRACE: *The Leader in Cloud Monitoring | Dynatrace*. <https://www.dynatrace.com/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Dyn20c] DYNATRACE: *What is OpenTelemetry? Everything you wanted to know*. <https://www.dynatrace.com/news/blog/what-is-opentelemetry/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Esp21] ESPERTECH: *Esper*. <https://espertech.com/esper>, 2021. – [Online; abgerufen am 09.01.2021]
- [Fac20a] FACEBOOK: *Instagram App Screenshot*. <https://www.instagram.com/>, 2020
- [Fac20b] FACEBOOK: *React - A JavaScript library for building user interfaces*. <https://reactjs.org>, 2020. – [Online; abgerufen am 12.10.2020]
- [Fil20] FILIPE, Ricardo Ângelo S.: *Client-Side Monitoring of Distributed Systems*, Universidade de Coimbra, Diss., 2020
- [Fre91] FREEDMAN, Roy S.: Testability of software components. In: *IEEE transactions on Software Engineering* 17 (1991), Nr. 6, S. 553–564
- [Fun20] FUNCTIONAL SOFTWARE: *About Sentry | Sentry*. <https://sentry.io/about/>, 2020. – [Online; abgerufen am 23.11.2020]
- [GB20] GREIF, Sacha ; BENITTE, Raphaël: *State of JS 2020: Front-end Frameworks*. <https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>, 2020. – [Online; abgerufen am 15.11.2020]
- [Goo20a] GOOGLE: *Angular*. <https://angular.io>, 2020. – [Online; abgerufen am 12.10.2020]
- [Goo20b] GOOGLE: *Protocol Buffers | Google Developers*. <https://developers.google.com/protocol-buffers/>, 2020. – [Online; abgerufen am 16.12.2020]
- [Gra20] GRAF, Michael: *Bedeutung von Telemetrie für den Software Development Life Cycle*. Hochschule Heilbronn, 2020

- [Hop06] HOPMANN, Alex: *The story of XMLHTTP*. <https://web.archive.org/web/20070623125327/http://www.alexhopmann.com/xmlhttp.htm>, 2006. – [Online; abgerufen am 27.10.2020]
- [Hou20] HOUND TECHNOLOGY: *Why Honeycomb - Honeycomb*. <https://www.honeycomb.io/why-honeycomb/>, 2020. – [Online; abgerufen am 23.11.2020]
- [Jae20a] JAEGER AUTHORS, The: *Architecture — Jaeger documentation*. <https://www.jaegertracing.io/docs/1.21/architecture/>, 2020. – [Online; abgerufen am 31.12.2020]
- [Jae20b] JAEGER AUTHORS, The: *Jaeger: open source, end-to-end distributed tracing*. <https://www.jaegertracing.io/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Jos19] JOSEPHSEN, Dave: iVoyeur: Distributive Tracing. In: *login*: 44 (2019), Nr. 4, S. 56. – ISSN 1044–6397
- [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
- [Log20a] LOGROCKET: *LogRocket / Logging and Session Replay for JavaScript Apps*. <https://logrocket.com/>, 2020. – [Online; abgerufen am 23.11.2020]
- [Log20b] LOGROCKET: *Performance*. <https://docs.logrocket.com/docs/performance>, 2020. – [Online; abgerufen am 16.12.2020]
- [Mic20a] MICROSOFT: *Microsoft 365 apps say farewell to Internet Explorer 11 and Windows 10 sunsets Microsoft Edge Legacy*. <https://techcommunity.microsoft.com/t5/microsoft-365-blog/microsoft-365-apps-say-farewell-to-internet-explorer-11-and/ba-p/1591666>, 2020. – [Online; abgerufen am 29.10.2020]
- [Mic20b] MICROSOFT: *New year, new browser – The new Microsoft Edge is out of preview and now available for download*. <https://blogs.windows.com/windowsexperience/2020/01/15/new-year-new-browser-the-new-microsoft-edge-is-out-of-preview-and-now-available-for-01.15>, 2020. – [Online; abgerufen am 29.10.2020]
- [MM20a] MOZILLA ; MITWIRKENDE individuelle: *console - Web APIs / MDN*. <https://developer.mozilla.org/en-US/docs/Web/API/Console>, 2020. – [Online; abgerufen am 19.10.2020]

- [MM20b] MOZILLA ; MITWIRKENDE individuelle: *Content Security Policy (CSP) - HTTP / MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP/>, 2020. – [Online; abgerufen am 15.10.2020]
- [MM20c] MOZILLA ; MITWIRKENDE individuelle: *Cross-Origin Resource Sharing (CORS) - HTTP / MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>, 2020. – [Online; abgerufen am 15.10.2020]
- [New20a] NEW RELIC: *Announcing OpenTelemetry Beta support in New Relic One - New Relic Blog*. <https://blog.newrelic.com/product-news/opentelemetry-beta-support-new-relic-one/>, 2020. – [Online; abgerufen am 20.11.2020]
- [New20b] NEW RELIC: *New Relic / Deliver more perfect software*. <https://opentelemetry.io/registry/>, 2020. – [Online; abgerufen am 20.11.2020]
- [OKSK15] OREN, Yossef ; KEMERLIS, Vasileios P. ; SETHUMADHAVAN, Simha ; KEROMYTIS, Angelos D.: The spy in the sandbox: Practical cache attacks in javascript and their implications. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, S. 1406–1418
- [OPBW06] OSHRY, Matt ; PORTER, Brad ; BODELL, Michael ; W3C, World Wide Web C.: *Authorizing Read Access to XML Content Using the <?access-control?> Processing Instruction 1.0*. <https://www.w3.org/TR/2006/WD-access-control-20060517/>, 2006. – [Online; abgerufen am 27.10.2020]
- [Ope20a] OPENCENSUS: *OpenCensus*. <https://opencensus.io/introduction/#overview>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope20b] OPENTELEMETRY: *About / OpenTelemetry*. <https://opentelemetry.io/about/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope20c] OPENTELEMETRY: *opentelemetry-specification/unified-collection.png*. <https://github.com/open-telemetry/opentelemetry-specification/blob/8e7b2cc17f2c572282c4e5e4d3cc54401749d8ff/specification/logs/img/unified-collection.png>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ope20d] OPENTELEMETRY: *Registry / OpenTelemetry*. <https://opentelemetry.io/registry/>, 2020. – [Online; abgerufen am 20.11.2020]
- [Ope20e] OPENTELEMETRY: *Write guidelines and specification for logging libraries to support OpenTelemetry-compliant logs*. <https://github.com/open-telemetry/opentelemetry-specification/issues/894>, 2020. – [Online; abgerufen am 19.11.2020]

- [Ope20f] OPENTRACING: *The OpenTracing Semantic Specification*. <https://github.com/opentracing/specification/blob/c064a86b69b9d170ace3f4be7dbacf47953f9604/specification.md>, 2020. – [Online; abgerufen am 11.12.2020]
- [Ope20g] OPENTRACING: *What is Distributed Tracing?* <https://opentracing.io/docs/overview/what-is-tracing/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ora20] ORACLE: *Java Debug Wire Protocol*. @<https://download.java.net/java/GA/jdk14/docs/specs/jdwp/jdwp-spec.html>, 2020. – [Online; abgerufen am 23.10.2020]
- [OS⁺18] ORIOL, Marc ; STADE, Melanie ; ; FOTROUSI, Farnaz ; NADAL, Sergi ; VARGA, Jovan ; SEYFF, Norbert ; ABELLO, Alberto ; FRANCH, Xavier ; MARCO, Jordi ; SCHMIDT, Oleg: FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring. In: *2018 IEEE 26th International Requirements Engineering Conference (RE)* IEEE, 2018, S. 217–227
- [Pow06] POWERS, Shelley: *Learning JavaScript*. O'Reilly Media Inc., 2006 (Java Series). – ISBN 9780596527464
- [Pro20] PROMETHEUS AUTHORS, The: *Prometheus - Monitoring system & time series database*. <https://prometheus.io/>, 2020. – [Online; abgerufen am 19.11.2020]
- [Ran09] RANGANATHAN, Arun: *cross-site xmlhttprequest with CORS*. <https://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/>, 2009. – [Online; abgerufen am 27.10.2020]
- [Sch20] SCHENCK, Don: *Istio Tracing & Monitoring: Where Are You and How Fast Are You Going?* <https://developers.redhat.com/blog/2018/04/03/istio-tracing-monitoring/>, 2020. – [Online; abgerufen am 17.12.2020]
- [SD13] SHREATEH, Khalil ; DEWEY, Caitlyn: Mark Zuckerberg's Facebook page was hacked by an unemployed Web developer. In: *The Washington Post* (2013), Aug. – [Online; abgerufen am 21.12.2020]
- [Sen20a] SENTRY: *getsentry/sentry-javascript: Official Sentry SDKs for Javascript*. <https://github.com/getsentry/sentry-javascript>, 2020. – [Online; abgerufen am 23.11.2020]
- [Sen20b] SENTRY: *Self-Hosted Sentry | Sentry Developer Documentation*. <https://develop.sentry.dev/self-hosted/>, 2020. – [Online; abgerufen am 23.11.2020]

- [Shk20] SHKURO, Yuri: *Take OpenTracing for a HotROD ride*. <https://medium.com/opentracing/take-opentracing-for-a-hotrod-ride-f6e3141f7941>, 2020. – [Online; abgerufen am 17.12.2020]
- [SM12] SUMARAY, Audie ; MAKKI, Shamila K.: A comparison of data serialization formats for optimal efficiency on a mobile platform. In: *Proceedings of the 6th international conference on ubiquitous information management and communication*, 2012, S. 1–6
- [Sma19] SMARTBEAR SOFTWARE: *Gherkin Syntax*. <https://cucumber.io/docs/gherkin/>, 2019. – [Online; abgerufen am 14.11.2020]
- [Spl20] SPLUNK: *The Data-to-Everything Platform Built for the Cloud | Splunk*. <https://www.splunk.com/>, 2020. – [Online; abgerufen am 24.11.2020]
- [Sri18] SRIDHARAN, Cindy: *Distributed Systems Observability: A Guide to Building Robust Systems*. O'Reilly Media Inc., 2018. – ISBN 9781492033424
- [Sta20] STATISTA: *Global market share held by leading desktop internet browsers from January 2015 to June 2020*. <https://www.statista.com/statistics/544400/market-share-of-internet-browsers-desktop/>, September 2020. – [Online; abgerufen am 29.10.2020]
- [STM+20a] 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*, 2020, S. 85–96
- [STM+20b] 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
- [SUP20] SUPERSEDE: *EU Project SUPERSEDE will seek to improve users' quality of experience when using software applications*. <https://cordis.europa.eu/article/id/129515-eu-project-supersede-will-seek-to-improve-users-quality-of-experience-when-using-software-app/de>, 2020. – [Online; abgerufen am 11.01.2021]
- [The20] THE LINUX FOUNDATION: *Kubernetes Components*. <https://kubernetes.io/docs/concepts/overview/components/>, 2020. – [Online; abgerufen am 14.11.2020]
- [W3C06] W3C, World Wide Web C.: *The XMLHttpRequest Object*. <https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>, 2006. – [Online; abgerufen am 27.10.2020]

- [W3C20] W3C, World Wide Web C.: *About W3C Standards*. <https://www.w3.org/standards/about.html>, 2020. – [Online; abgerufen am 29.10.2020]
- [YM20] YOU, Evan ; MITWIRKENDE individuelle: *Vue.js*. <https://vuejs.org/>, 2020. – [Online; abgerufen am 12.10.2020]
- [ZE11] ZIKOPOULOS, Paul ; EATON, Chris: *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011. – 5 S. – ISBN 9780071790543