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, 9. Dezember 2020

Kurzfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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, consectetuer 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, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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, consectetuer 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 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 [**?**]. 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 bpsw. 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 Verständnisgewinnung zu Interaktionen eines Nutzers und des Verhaltens der Applikation. Des Weiteren wird die Nachvollziehbarkeit als solche formal beschrieben und was sie im Projektumfeld 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 mit Hilfe 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 [?].

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

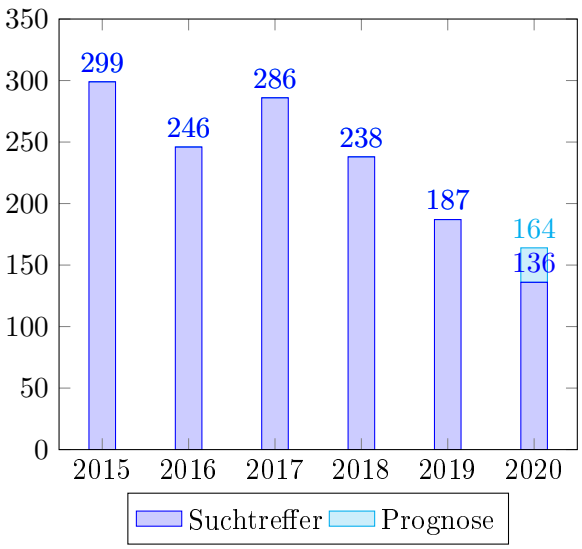

Abb. 2.1: Studien zur Browserkompatibilität, eigene Darstellung (vgl. 7.1)

Explorer, den Microsoft Edge Browser, von einer proprietären Browserengine zu Chromium gewechselt [?] und enthält somit den selben Kern wie Chrome und Opera. Zum Ende 2020 wird zudem der Support für den Internet Explorer 11 eingestellt [?]. 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 [?].

2.1.2 JavaScript

Als JavaScript 1997 veröffentlicht und in den NetScape Navigator integriert wurde, gab es die berechtigen 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. [?]

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 [?]. 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 [?] fand die Methode Anklang und Einsatz bei Entwicklern und ist seitdem der Grundstein für unser dynamisches und interaktives Web [?].

Webapplikationen wurden nun immer beliebter, aber Entwickler klagten darüber, dass Browser die Abfragen von JavaScript nur auf dem bereitstellenden Webserver, also "same-origin", erlauben[?]. Im selben Jahr der Standardisierung von Ajax, wurde ein erster Entwurf zur Ermöglichung und Absicherung von Abrufen domänenfremder Ressourcen eingereicht [?], 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, und 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 [**?**], React [**?**] oder Vue.js [**?**].

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 Seitenabrufe notwendig sind [**?**].

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 [**?**].

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 [**?**]. Nach Freedman [**?**] und Scrocca *et al*[**?**] lässt sich diese Definition auch auf Softwaresysteme übertragen.

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

2.3.1 Nutzen

Tritt beispielsweise ein Softwarefehler (Bug) bei einem Nutzer auf, aber die Stakeholder erhalten nicht ausreichende Informationen, so kann der Bug ignoriert werden oder gering priorisiert und in Vergessenheit geraten. Dies geschah im Jahr 2013, als Khalil Shreateh eine Sicherheitslücke bei Facebook fand und diesen bei Facebooks Bug-Bounty-Projekt Whitehat meldete [**?**]. 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 Shreateh 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 Singe-Page-Applications besondere Barrikaden, die es den Stakeholdern 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 [?].

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 standardmäßig[1]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.

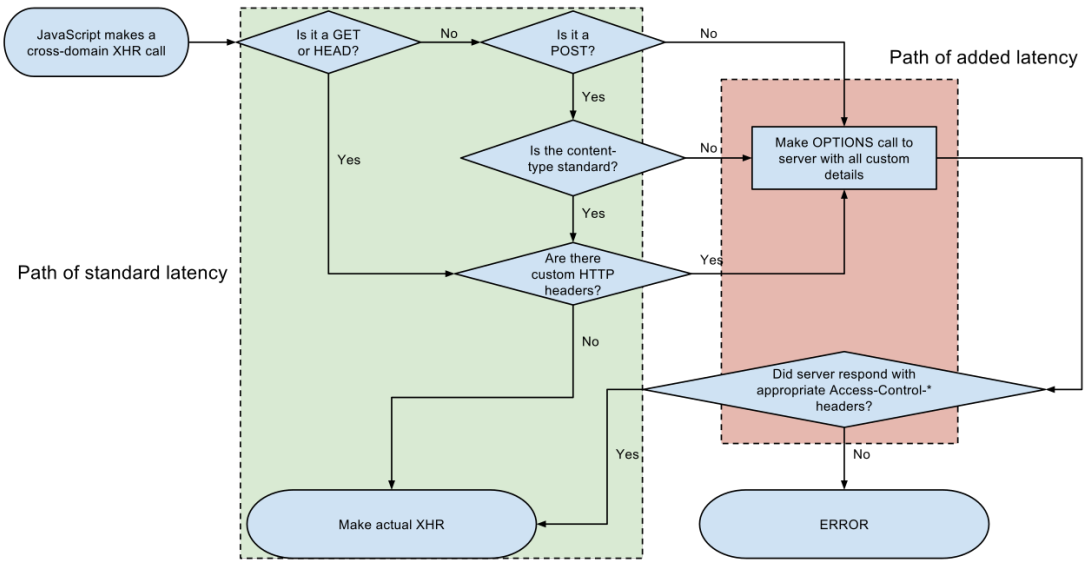

Abb. 2.2: Flowchart über den Ablauf von Ajax-Anfragen mit CORS [?]

[1]Standardmäßig ist eine Anfrage, wenn 1. die Methode GET, HEAD oder POST entspricht; 2. keine eigene Headern 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" [?].

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 [**?**]. 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 [**?**]. 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 beschrieben 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 die Stakeholder gelangen.

2.3.5 Fernzugriff

Ein weiterer Punkt, der die Umgebung “Browser“ von anderen unterscheidet, ist dass die Stakeholder sich normalerweise nicht auf die Systeme der Nutzer schalten können. Bei Expertenanwendungen 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 [**?**], welches Java unterstützt.

3 Methoden und Praktiken

In diesem Kapitel soll beschrieben werden, wie eine Nachvollziehbarkeit in Webapplikationen erreicht werden kann. Spezielle Methoden und Praktiken sollen vorgestellt und beleuchtet werden.

3.1 Methoden

3.1.1 Logging

Folgende Fragen sollen zur Methode beantwortet werden

1. *Gibt es Besonderheiten zu Logging in anderen Projekten (Backend vs. Frontend)?*
2. *Wie können Logs an einen auswertenden Stakeholder gelangen?*
3. *Welches Verhalten kann hiermit aufgedeckt/nachvollziehbar gemacht werden?*

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. Ein Zähler, der die Anzahl an Netzwerkanfragen zählt
3. 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 so unerwünschte Abweichungen aufdecken.

3.1.3 Tracing

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

Beschreibung von OpenTracing und der Bedeutung von Spans und Traces

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 schief gelaufen ist (vgl Abbildung 3.1). Hiermit können Fehler, aber auch unverständliche Workflows, aufgedeckt werden. Weiterhin kann die Intention des Nutzers ermittelt werden, vorausgesetzt er gibt dies an.

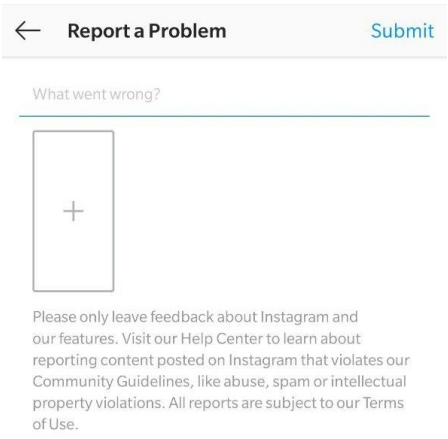

Abb. 3.1: Fehlerbericht in der Instagram App [?]

Bettenburg *et al*[?] 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.

3.2 Praktiken aus der Fachpraxis

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 einige Praktiken entwickelt, die diese Technologien benutzen, um ihr Ziel zu erreichen. Diese Praktiken werden folgend vorgestellt.

3.2.1 System Monitoring

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

3.2.2 Log Management

Log Management umfasst die Erfassung, Speicherung, Verarbeitung und Analyse von Logdaten von Anwendungen. Weiterhin bieten Werkzeuge hierbei oftmals fundierte Such- und Meldefunktionen. Zur Behebung der Hürde in ein bestehendes System zu integrieren, werden oftmals eine Vielzahl von Integrationen für gängige Frameworks und Bibliotheken angeboten.

3.2.3 Application Performance Monitoring (APM)

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

3.2.4 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.

3.2.5 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.

3.2.6 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.

3.2.7 Session-Replay

Session-Replay bedeuted, 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.

3.3 Werkzeuge und Technologien

3.3.1 Fachpraxis

Bei der Recherche zu Werkzeugen und Technologien aus der Fachpraxis wurden eine Handvoll 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 dieser Recherche wurde immer wieder auf den Standard OpenTelemetry gestoßen, weshalb dieser zunächst kurz beschrieben wird.

3.3.1.1 OpenTelemetry

OpenTelemetry (OTel) [?] ist ein sich derzeit entwickelnder herstellerunabhängiger Standard, um Tracing-, Metrik- und Logdaten[1]zu erfassen, verarbeiten, analysieren und zu visualisieren. Der Standard fasst die beiden Standards OpenTracing und OpenCensus [?] zusammen und hat sich als Ziel gesetzt diese zu erweitern. Hinter dem Standard stehen u. A. die Cloud Native Computing Foundation (CNCF), Google, Microsoft, und führende Hersteller in Tracing und Monitoring-Lösungen. Ein erster Release ist für Ende 2020/Anfang 2021 geplant. Ziel ist es, dass Entwickler Tools 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 angesprochen werden können. Die technische Infrastruktur einer auf OTel basierenden Lösung kann in Abbildung 3.2 betrachtet werden. Im groben definiert OTel folgende Komponenten: API, SDK, Exporter, Collector und Backend (vgl. Abbildung 3.3).

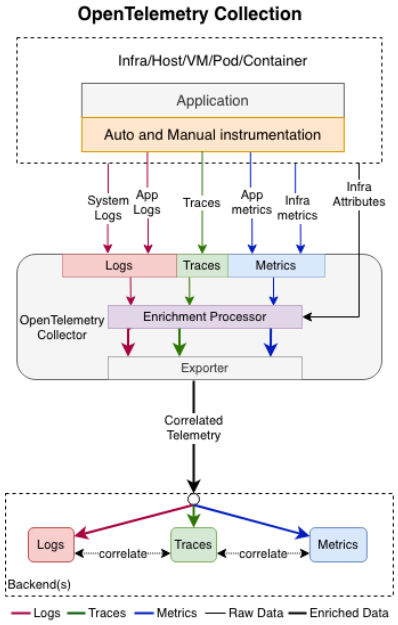

Abb. 3.2: Schaubild einer Lösung auf Basis von OTel [?]

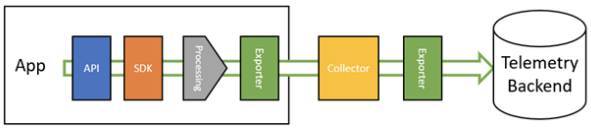

Abb. 3.3: OTel Komponenten [?]

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

3.3.1.2 New Relic

New Relic [?] 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 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, weiterhin 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 [?]. 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 [?].

3.3.1.3 Dynatrace

Dynatrace [?] 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 die selben Informationen dar wie New Relic, wobei Dynatrace das Ganze visuell ansprechender darstellt. Dynatrace bietet auch die Funktionalität vom 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 [**?**]. Eine Integration des Dienstes Dynatrace ins Ökosystem von OTel gibt es jedoch noch nicht.

3.3.1.4 Sentry

Sentry [**?**] ist ein SaaS Produkt der Functional Software Inc, welches 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 [**?**], 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 die ganzheitliche Nachvollziehbarkeit, aber dafür sind die gesammelten Fehlerinformationen zahlreich und helfen beim Nachvollziehen.

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

3.3.1.5 LogRocket

LogRocket [?] 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. Mit Hilfe der gesammelten Daten wird die gesamte Sitzung des Nutzers nachgestellt, hierbei ist die Anwendung, den Nutzer und seine Aktionen, die Netzwerkaufrufe sowie das DOM zu sehen. Die Nachstellung wird video-ähnlich aufbereitet und erlaubt ein präzises Nachvollziehen der zeitlichen Reihenfolge und Bedeutung (vgl. Abbildung 3.4).

Neben dem JavaScript SDK bietet LogRocket quelloffenene Plugins für folgende Bibiotheken: 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.

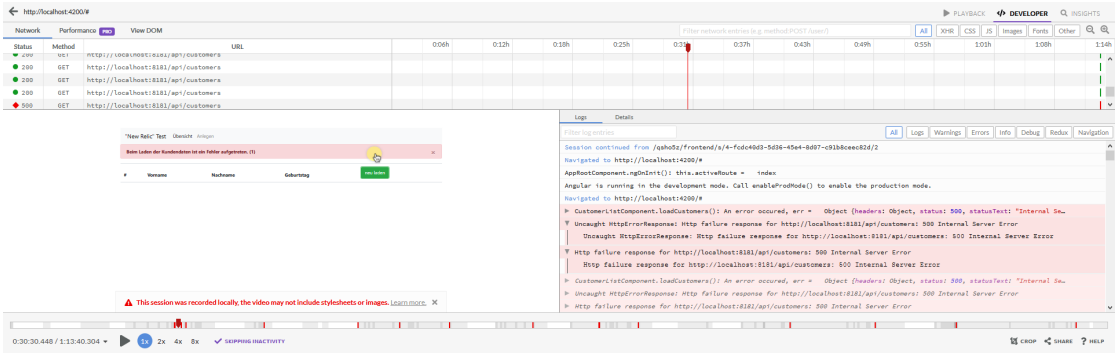

Abb. 3.4: Beispiel eines Session-Replays bei LogRocket

3.3.1.6 Splunk

Splunk [?] ist ein Softwareprodukt und ein SaaS des gleichnamigen Unternehmens. Splunk fungiert als eine universelle Datensenke, z. B. für Metriken, Logdaten sowie andere Daten. Splunk bietet Funktionen diese Daten zu durchsuchen, überwachen, analysieren und visualisieren. Splunk wird 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 angehangen werden.

An Splunk wurden über dieselbe Schnittstelle Log- und Errorinformationen gesendet. Hierbei konnte Splunk gut das Logmanagement erfüllen, denn Logeinträge lassen 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 Wasserfalldiagrammen fehlt gänzlich. Hiermit lassen sich keine spezialisierten Tracing-Werkzeuge ersetzen.

3.3.1.7 Honeycomb

Honeycomb [?] 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 [?]. Es ist ein System für verteiltes Tracing von der Datensammlung bis hin zur Visualisierung, es unterstützt und implementiert den Standard OpenTracing. Eine Unterstützung des OpenTelemetry Standards

ist derzeit im Gange. Weiterhin kann Jaeger dazu benutzt werden, Metriken nach Prometheus [**?**] zu exportieren, einem weiteren CNCF Projekt zur Speicherung und Visualisierung von Daten.

Jaeger wurde nicht aufgesetzt, sondern über Dokumentation und Kommunikation mit Kollegen der Open Knowledge evaluiert. Wie zuvor bei Honeycomb beschrieben, beschränkt sich Jaeger sehr auf Tracing, aber kann sich dafür auf dieses Gebiet spezialisieren und bietet eine gesamte Infrastruktur.

3.3.1.9 Weiteres

Bei meiner 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: Google Cloud Trace, Zipkin

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

3.3.2 Literatur

1. *Gibt es hierzu Ansätze in der Literatur?*
2. *Wie sehen diese aus, welchen Zweck erfüllen sie?*
3. *Sind sie vergleichbar mit Ansätzen aus der Fachpraxis?*

4 Erstellung Proof-of-Concept

4.1 Anforderungen

4.1.1 Definitionen

Um die Anforderungen systematisch zu kategorisieren, werden folgend zwei Modelle vorgestellt, mit denen die Anforderungen kategorisiert werden.

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

Kürzel	Titel	Beschreibung
B	Basismerkmal	Merkmale, die als selbstverständlich angesehen werden. Eine Erfüllung erhöht kaum die Zufriedenheit, jedoch eine Nichterfüllung führt zu starker Unzufriedenheit
L	Leistungs-merkmal	Merkmale, die der Kunde erwartet und bei nicht Vorhandensein in Unzufriedenheit äußert. Ein Vorhandensein erzeugt Zufriedenheit, beim übertreffen umso mehr.
S	Begeisterungs-merkmal	Merkmale, mit der man sich von der Konkurrenz herabsetzen kann 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	Rückweisungs-merkmal	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 [**?**] 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- sowie 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

Wie werden die Anforderungen erhoben?

Die primäre Quelle von Anforderungen, stellen 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.

Kürzel	Titel	Beschreibung
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 referenziert werden kann.

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
10	nan	B	f	A
nan				

4.1.3.2 Funktionsumfang

Id	Name	Kano-Modell	Funktionsart	Quelle
1010	Schnittstellen-Logging	B	f	S
Das Aufrufen von Schnittstellen soll mittels einer Logmeldung notiert werden. Hierbei sollen relevante Informationen wie Aufrufparameter mit notiert werden.				

Id	Name	Kano-Modell	Funktionsart	Quelle
1011	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
1020	Fehler-Monitoring	B	f	S
Tritt ein Fehler auf, der nicht gefangen wurde, so soll dieser automatisch erfasst werden und um weitere Attribute ergänzt werden. Sonstige Fehler können auch erfasst werden, aber hierbei Bedarf es einem manuellen Aufruf einer Schnittstelle				

Id	Name	Kano-Modell	Funktionsart	Quelle
1100	Logübertragung	B	f	S
Ausgewählte Logmeldungen sollen an ein Partnersystem weitergeleitet werden. Die Auswahl könnte über die Kritikalität, also dem Log-Level, der Logmeldung erfolgen.				

Id	Name	Kano-Modell	Funktionsart	Quelle
1101	Fehlerübertragung	B	f	S
Sämtliche erfassten Fehler sollen an ein Partnersystem weitergeleitet werden.				

4.1.3.3 Eigenschaften

Id	Name	Kano-Modell	Funktionsart	Quelle
2010	Resilienz der Übertragung	S	f	S
Daten die der Nachvollziehbarkeit dienen, sollen wenn möglich bei einer fehlgeschlagenen Verbindung nicht verworfen werden. Sie sollen mindestens 120s vorgehalten werden und in dieser Zeit sollen wiederholt Verbindungsversuche unternommen werden.				

Id	Name	Kano-Modell	Funktionsart	Quelle
2020	Batchverarbeitung	S	f	S
Daten, die der Nachvollziehbarkeit dienen, sollen wenn möglich gruppiert an externe Systeme gesendet werden. Hierbei ist eine kurze Aggregationszeit von bis zu 10s akzeptabel.				

4.1.3.4 Partnersysteme

Id	Name	Kano-Modell	Funktionsart	Quelle
4010	Partnersystem "Log-Management"	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
4020	Partnersystem Ërror-Monitoring"	B	f	A+S
Es existiert ein Partnersystem, zu dem Fehler weitergeleitet werden. Dieses System soll die Fehler speichern und den Entwicklern und Betreibern eine Einsicht in die erfassten Fehler bieten.				

Id	Name	Kano-Modell	Funktionsart	Quelle
4021	Visualisierung Ërror-Monitoring"	L	f	A+S
Das Partnersystem, zu dem die Fehler weitergeleitet werden, soll diese grafisch darstellen können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
4021	Alerting Ërror-Monitoring"	S	f	A+S
Das Partnersystem, zu dem die Fehler weitergeleitet werden, soll bei Auftreten von bestimmten Fehlern oder Fehleranzahlen eine Meldung erzeugen (per E-Mail, Slack, o. Ä.).				

Id	Name	Kano-Modell	Funktionsart	Quelle
4030	Partnersystem "Tracing"	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
4031	Visualisierung "Tracing"	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
4030	Partnersystem "Metriken"	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
4031	Visualisierung "Metriken"	L	f	A+S
Das Partnersystem, zu dem die Metriken weitergeleitet werden, soll diese grafisch darstellen können.				

Id	Name	Kano-Modell	Funktionsart	Quelle
4032	Alerting "Metriken"	S	f	A+S
Das Partnersystem, zu dem die Metriken weitergeleitet werden, soll bei Auftreten von bestimmten Metrikwerten oder Überschreitungen von Schwellen eine Meldung erzeugen (per E-Mail, Slack, o. Ä.).				

TODO: what about structured/unstructed logs?

4.2 Vorstellung der Demoanwendung

In diesem Abschnitt soll die Demoanwendung vorgestellt werden, anhand dessen das Proof-of-Concept erstellt wird. Damit das Proof-of-Concept erstellt werden kann, muss die Demoanwendung die zuvor beschriebenen Probleme aufweisen, hierbei sollen die Probleme möglichst realitätsnah sein und nicht frei erfunden.

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 des 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 um festzustellen wie lange sich Nutzer auf einer Seite aufhalten. 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. eine 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. Bei könnten wichtige Verarbeitungsmethoden auch 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 bei Logdaten. 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 gesetzt 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 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 die u. A. 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-Wasserfallgraph 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 Komponente verwenden. Dieser Aufbau lässt sich in Abbildung 4.1 betrachten.

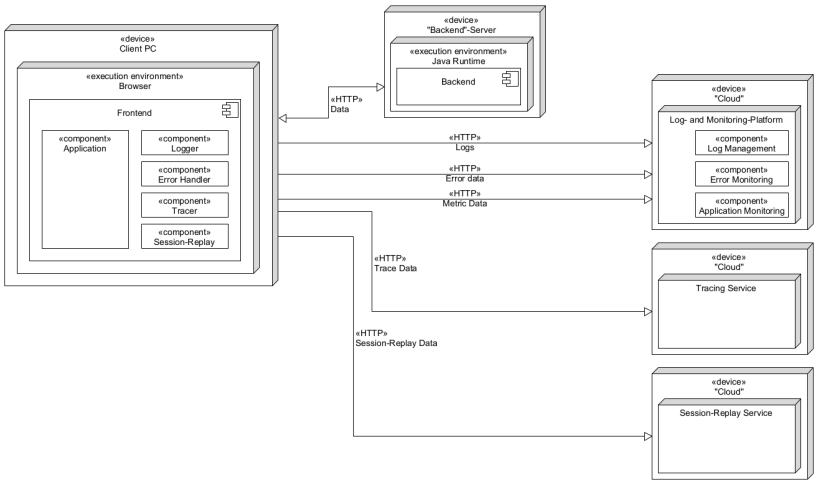

Abb. 4.1: Grobe Architektur

Wie in der Datenerfassung erwähnt, werden die einzelnen Datentypen unterschiedlich erhoben und besitzen somit auch eine andere Eigenschaften. Wie IBM bei Big Data definiert [**?**], 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 gewisse 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 Menge an Informationen. Der Aspekt des Volumes ist zudem auch sehr wichtig und eine Visualisierung dessen für das vorhergehende Konzept findet sich in Abbildung 4.2.

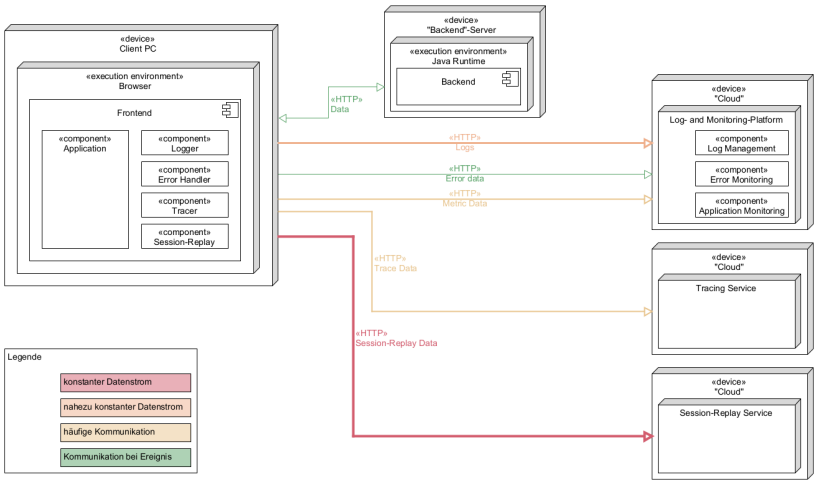

Abb. 4.2: Grobe Architektur mit Datendurchsatz

4.3.3 Technologie-Stack

Welche Technologien werden eingesetzt, um die Architektur umzusetzen und zu erfüllen?

4.3.4 Übertragbarkeit

Eignet sich das Konzept in Hinsicht auf Übertragbarkeit auf andere Softwareprojekte?

Dieser Punkt wird in Kapitel 5 erneut betrachtet und es wird versucht die tatsächliche Übertragbarkeit der erstellten Lösung zu erörtern.

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, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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 Browserkompatiblität dargestellt. Die Daten hierfür wurden über die Literatursuchmaschine “Google Scholar“ am 29.10.2020 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. Der Prognosewert fürs Jahr 2020 wurde pauschal so berechnet, dass in den letzten zwei Monaten proportional gleich viele Veröffentlichungen erscheinen, also: $\left\lceil \frac{136*12}{10} \right\rceil$

Jahr	Treffer
2015	299
2016	246
2017	286
2018	238
2019	187
2020	136
2020 (Prognose)	164

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

CSP Content-Security-Policy

OTel OpenTelemetry

PoC Proof-of-Concept

SaaS Software-as-a-Service

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

W3C World Wide Web Consortium

XHR XMLHttpRequest

XSS Cross-Site-Scripting

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis