Guidelines für die Integration von User Interfaces in Microservice-basierten Systemen

MASTER THESIS

Pascal Vahldiek

Eingereicht am 13. Dezember 2022

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open-Source-Software

Betreuer:
Georg Schwarz
Prof. Dr. Dirk Riehle, M.B.A.

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 13. Dezember 2022

License

Erlangen, 13. Dezember 2022

Abstract

Microservice-based systems consist of loosely coupled, independent distributed services, communicating over the network.

The integration of the frontend in microservices is a central problem. In contrast to the backend services, one independent service is often used for the frontend of an application. This contradicts the principle of microservice-based systems, since a central, monolithic service is responsible for the entire frontend. This thesis examines how frontend user interfaces can be integrated into microservices. The exisiting frontend architectures in a microservice context are presented and analyzed with regard to advantages, disadvantages and challenges.

A structured literature analysis is carried out to collect data for theory formation. In addition, expert interviews are conducted as part of a case study with industrial partners in order to get an overview of the methods used in practice. Subsequently, in the scope of an action research study, one integration options for user interfaces in microservices is applied for a specific application. The results of the action research are evaluated and compared to the contents of the structured literature analysis. The objective of the thesis is to show the implications of the comparison with a guideline.

Practitioners can use this comparison to select a suitable frontend integration solution for their application. Scientists can build on this work to develop further integration solutions, refine existing ones, and extend them.

Zusammenfassung

Microservice-basierte Systeme setzen sich aus einzelnen, eigenständigen Services zusammen, die über Netzwerkmechanismen kommunizieren.

Eine zentrale Problemstellung bei Microservices ist das Frontend. Anders als bei Backend Services, wird in der Praxis für das Frontend einer Anwendung häufig ein eigenständiger Service verwendet. Dies führt den Gedanken der domänengesteuerten Modularisierung nicht zu Ende, da das Frontend nicht einbezogen wird. In dieser Arbeit wird untersucht, wie User Interfaces im Frontend in Microservices integriert werden können. Die verschiedenen Architekturen bei der Umsetzung werden herausgearbeitet und hinsichtlich Vorteilen, Nachteilen und Herausforderungen analysiert.

Für die Datenerfassung der Theoriebildung wird eine strukturierte Literaturanalyse durchgeführt. Zusätzlich werden Experteninterviews im Rahmen einer Case Study mit Industriepartnern geführt, um einen Überblick über die in der Praxis eingesetzten Verfahren zu erhalten. Im Anschluss daran werden mithilfe von Aktionsforschung verschiedene Integrationsmöglichkeiten für User-Interfaces in Microservices bei einer konkreten Anwendung entwickelt. Die Ergebnisse der Aktionsforschung werden evaluiert und den Inhalten der strukturierten Literaturanalyse gegenübergestellt. Ziel der Arbeit ist es mit einer Richtlinie die Implikationen der Gegenüberstellung aufzuzeigen.

Praktiker können diese Gegenüberstellung nutzen, um für ihren Anwendungsfall eine geeignete Frontend-Integrationslösung zu wählen. Wissenschaftler können auf dieser Arbeit aufbauen, um weitere Integrationslösungen zu erarbeiten, bestehende zu verfeinern, und zu erweitern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Acronyms

SOA Service-orientierte Architektur

REST Representational State Transfer

ETL Extract-Transform-Load

CI Continuous Integration

MVC Model-View-Controller

SCS Self-Contained-System

CRUD Create, Read, Update, Delete

DOM Domain Object Model

UX User Experience

BFF Backend for Frontends

API Application Programming Interface

AMQP Advanced Message Queuing Protocol

DDD Domain-Driven Design

UML Unified Modeling Language

SPA Single Page Application

1 Aufbau der Arbeit

Die Arbeit ist in zwei übergreifende Abschnitte gegliedert. Der erste Teil umfasst eine kurzgefasste Ausarbeitung der Theorie und Ergebnisse. Die Gestaltung des Teils orientiert sich an dem Aufbau eines wissenschaftlichen Artikels. Im zweiten Abschnitt werden weiterführende Erläuterungen dargelegt, die zum allgemeinen Verständnis dienen sollen. In diesem Teil werden Konzepte und verwendete Technologien vertieft. Die weiterführenden Erläuterungen dienen als Ergänzung der in der Theorie dargestellten Themen, weshalb an passenden Passagen Verweise zwischen den beiden Kapitel existieren.

Kapitel 2 beinhaltet die Theoriebildung. Die Einleitung in Kapitel 2.1 führt auf das Thema der Arbeit hin und definiert die zugrunde liegenden Forschungsfragen. Nach einem Überblick über themenverwandte Arbeiten, wird in Kapitel 2.3 der Forschungsansatz erläutert. Der Abschnitt umfasst grundlegende Erklärungen zu den verwendeten Forschungsmethoden und deren Umsetzung. Anschließend werden in Kapitel 2.4 sowohl die Erkenntnisse aus Literaturanalyse als auch der Case Study in einer tabellarischen Form präsentiert. Die dargestellten Tabellen beziehen sich auf die in der Einleitung definierten Forschungsfragen. Daraufhin wird in Kapitel 2.5 die durchgeführte Aktionsforschung erläutert. Darin wird die Konzeption der Lösung und deren Implementierung vorgestellt. Abschließend werden die gewonnenen Erkenntnisse tabellarisch aufbereitet und mit den Erkenntnissen aus Literaturanalyse und Case Study abgeglichen. Zuletzt werden die Ergebnisse diskutiert und ein Ausblick auf die weitere Forschung gegeben.

In den weiterführenden Erläuterungen werden die Inhalte aus der Theoriebildung vertieft. Der erste Teil in Kapitel 3.1 erläutert das Konzept der Microservices und deren Modellierung. Abschnitt 3.2 konzentriert sich auf die in der Aktionsforschung verwendeten Werkzeuge und Technologien. Abschließend sind im Anhang weitere Grafiken und Auswertungen zu finden.

2 Kurzgefasste Ausarbeitung

2.1 Einleitung

Microservice-basierte Systeme setzen sich aus einzelnen, eigenständigen Services zusammen, die über Netzwerkmechanismen miteinander kommunizieren.

Die Dienste sind weitestgehend voneinander entkoppelt, werden nach Funktionalität separiert und stellen damit einen Ansatz für eine Service-orientierte Architektur (SOA) dar (Newman, 2021). Microservices erlauben eine langfristige und skalierbare Entwicklung des Systems und können unabhängig voneinander entwickelt, betrieben und getestet werden (Thönes, 2015).

Eine zentrale Problemstellung bei Microservices ist das Frontend. Anders als bei Backend Services, wird in der Praxis für das Frontend einer Anwendung häufig ein eigenständiger Service verwendet. Dies widerspricht dem Prinzip von Microservice-basierten Systemen, da dadurch ein zentraler, monolithischer Service für das gesamte Frontend verantwortlich ist. In dieser Arbeit wird untersucht, wie User Interfaces im Frontend in Microservices integriert werden können. Insbesondere werden die verschiedenen Architekturen bei der Umsetzung herausgearbeitet und hinsichtlich Vorteilen, Nachteilen und Herausforderungen analysiert.

Die Arbeit behandelt die folgende übergeordnete Forschungsfrage schwerpunktmäßig:

- **RQ:** Wie können User Interfaces in Microservices integriert werden?

Daraus leiten sich nachfolgende untergeordnete Forschungsfragen ab:

- **RQ1:** Welche verschiedenen Ansätze für die Integration von User Interfaces in Microservice-Architekturen gibt es?
- **RQ2:** Welche Vorteile bieten die unterschiedlichen Ansätze?
- **RQ3:** Welche Nachteile ergeben sich bei den verschiedenen Ansätzen?

- **RQ4:** Welche Herausforderungen gibt es bei der Frontend-Integration in Microservice-Architekturen?
- **RQ5:** Welche Richtlinien existieren für die Frontend-Integration in Microservice-Architekturen?

Die Forschungsfragen werden sowohl in einer strukturierten Literaturanalyse als auch in einer Case Study betrachtet und beantwortet. Außerdem wird eine Aktionsforschung durchgeführt, um die Erkenntnisse zu evaluieren und weitere Erfahrungen zu generieren.

2.2 Themenverwandte Arbeiten

Die Publikation (Harms et al., 2017) erweist sich als themenverwandte Arbeit. Das Ziel der Arbeit nähert sich dem Ziel dieser Arbeit an. Es thematisiert Vorteile und Nachteile unterschiedlicher Architekturen für die Frontend Integration in Microservices und definiert zugehörige Guidelines. Jedoch wurde in (Harms et al., 2017) die Datenerfassung im Rahmen explorativer Prototyp-basierter Forschung betrieben. Dies unterscheidet sich zu dieser Arbeit, da sowohl eine strukturierte Literaturanalyse als auch eine Case Study durchgeführt wurde. Außerdem werden in den Werken häufig nur einzelne Architekturen behandelt, aber keine Vergleiche zu anderen hergestellt. Die Literaturanalyse zeigt auch, dass es keine Werke gibt, die die Migration hin zu anderen Strategien behandeln.

2.3 Forschungsansatz

Für die Datenerfassung der Theoriebildung wird eine strukturierte Literaturanalyse durchgeführt. Zusätzlich werden Experteninterviews im Rahmen einer Case Study mit Industriepartnern geführt, um einen Überblick über die in der Praxis eingesetzten Verfahren zu erhalten.

Die Daten-Triangulation aus Literaturanalyse und Experteninterviews bietet mehrere Vorteile. Dadurch können Forschungsfragen in einem breiteren, praktischeren Kontext beantwortet werden. Außerdem ermöglicht es mehrere Sichtweisen auf den Forschungsaspekt (Carter et al., 2014).

Im Anschluss daran werden mithilfe von Aktionsforschung verschiedene Integrationsmöglichkeiten für User-Interfaces in Microservices bei einer konkreten Anwendung am Beispiel evaluiert. Die Ergebnisse der Aktionsforschung werden abschließend den Inhalten der strukturierten Literaturanalyse gegenübergestellt.

Ziel der Arbeit ist es mit einer Richtlinie die Implikationen der Gegenüberstellung aufzuzeigen. Die Forschungsmethoden werden in den folgenden Kapiteln dargelegt.

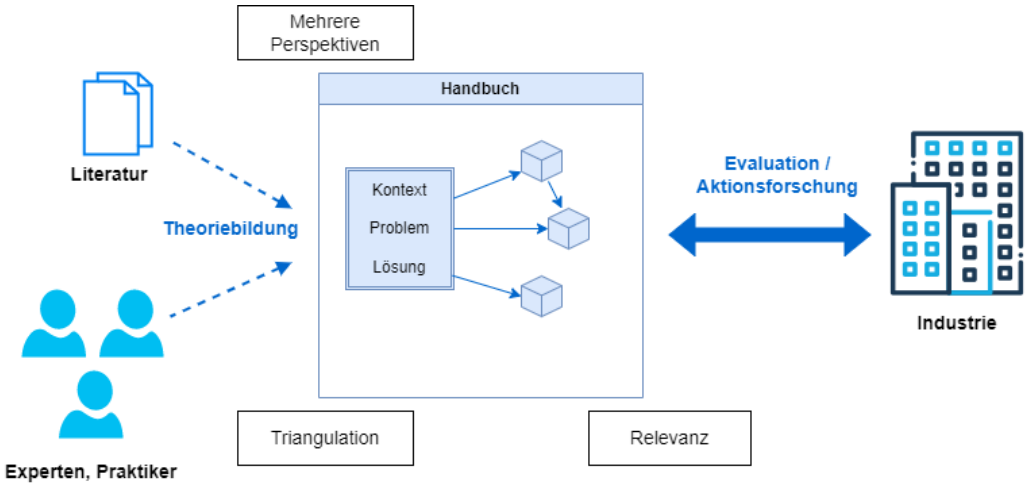

Abbildung 2.1: Forschungsansatz

2.3.1 Strukturierte Literaturanalyse

Um Erkenntnisse über die in Kapitel 2.1 auf Seite 3 definierten Forschungsfragen zu erlangen, wurde eine strukturierte Literaturanalyse durchgeführt. Diese orientiert sich an den von Kitchenham (2004) etablierten Richtlinien.

Suchprozess

Um geeignete Literatur einzugrenzen, wurde systematisch ein Suchbegriff entwickelt. Dieser erlaubt beliebige Kombinationen im Titel aus semantischen Relationen der Begriffe UI und Microservices. Ebenfalls werden Singular und Plural der Suchterme eingeschlossen. Tabelle 2.1 zeigt die Basis für die entwickelte Sucheingabe. Dauraufhin resultiert der folgende Suchbegriff:

```
allintitle: ("UI" OR "UIs" OR "User interface" OR "User interfaces"
    OR "Frontend" OR "Frontends" OR "Front End" OR "Front Ends")
AND ("microservice" OR "microservices"
    OR "micro service" OR "micro services")
```

Der Suchbegriff wird in der Datenbanken von Google Scholar, IEEExplore und ACM Digital Library angewendet.

Ein- und Ausschlusskriterien

Die Eignung einer Publikation wird auf Basis folgender Einschlusskriterien überprüft:

- Artikel, die in englischer Sprache angefertigt sind
- Artikel, die frei zugänglich sind
- Artikel, mit Themenfokus auf Frontends/UI (über den Suchbegriff)

UI (in title)	Microservices (in title)
UI	Microservice
UIs	Microservices
User interface	Micro service
User interfaces	Micro services
Frontend	
Frontends	

Tabelle 2.1: Suchprozess - Suchbegriffermittlung

- Artikel, mit Themenfokus auf Microservices (über den Suchbegriff)

Artikel, die dem folgenden Ausschlusskriterium unterlegen, werden nicht weiter berücksichtigt:

- Artikel, ohne Themenfokus auf Frontends/UI und Microservices

Nach dem Sammeln der Literatur über den Suchbegriff, werden geeignete Publikationen extrahiert. Der Auswahlprozess der Literatur läuft in drei Phasen ab:

1. Lesen des Abstracts und Ausschluss falls Themenfokus nicht relevant
2. Lesen des Ergebnis- oder Diskussionsteils der Publikation in unklaren Fällen
3. Lesen der gesamten Publikation, notieren relevanter Aspekte bezogen auf die Forschungsfragen

Qualitätsbewertung

Geeignete Publikationen sollen durch einen Peer-Review Prozess gelaufen sein. Dadurch werden Publikationen auf Konferenzen, Journale oder Workshop Ergebnisse eingeschränkt. White-Paper und Blog Artikel werden somit aus dem Suchprozess ausgeschlossen.

Snowballing

Snowballing ist ein weiteres Verfahren, um zusätzliche Literatur zu sammeln. Dabei wird zwischen Backward Snowballing und Forward Snowballing unterschieden. Backward Snowballing bezeichnet die Methode, bei der die von einer gewählten Literatur zitierten Werke gesammelt werden. Im Gegensatz dazu werden beim Forward Snowballing Werke aufgenommen, die eine gewählte Literatur zitieren. (Wohlin, 2014)

Backward und Forward Snowballing ergaben keine weiteren relevanten Werke.

Datensammlung

Bei den Publikationen, die den Auswahlprozess erfolgreich durchlaufen haben, werden die Quelle und eine Referenz darauf festgehalten. Zusätzlich werden Au-

tor, Forschungsmethoden, Anwendungsbereich und Themenfokus erfasst. Als Ergebnis der Datensammlung resultiert eine tabellarische Aufbereitung der Publikationen und zugehörigen Aspekte bezogen auf die in Kapitel 2.1 definierten Forschungsfragen **RQ1-RQ5**. Die aufbereitete Tabelle ist in Anhang B zu finden. Eine kurze Vorstellung der Literaturbasis und die Begründung für die Auswahl der Werke befindet sich im Kapitel der Tabelle.

Auswahl der Literatur

Wie in Kapitel 2.3.1 auf Seite 5 beschrieben, wird für die Literaturanalyse ein Suchbegriff entwickelt. Folgende Abbildung fasst den Such- und Auswahlprozess zusammen und zeigt die Anzahl der für jeden Schritt in Frage kommenden Werke auf.

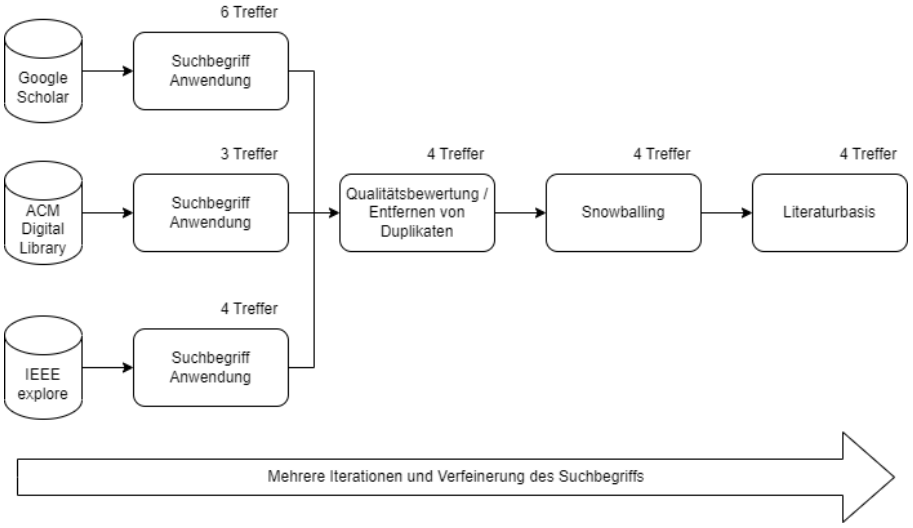

Abbildung 2.2: Auswahlprozess

Der Suchbegriff wird in der Datenbanken von Google Scholar, IEEEexplore und ACM Digital Library angewendet. Die Suche mit Google Scholar ergibt eine Anzahl von sechs potenziell relevanten Publikationen. Die Datenbanken IEEEexplore und ACM Digital Library ergeben keine weiteren Artikel, die mit Google Scholar nicht gefunden wird.

Nach Anwendung der auf Kapitel 2.3.1, Seite 6 beschriebenen Qualitätsbewertung, beschränkt sich die Anzahl der potenziell relevanten Werke auf vier.

Zweite Iteration

Die erste Analyse der Artikel zeigt auf, dass in der Literatur drei Frontend-Architekturen vorherrschend sind. Die dabei herausgearbeiteten Strategien **(RQ1)** werden daraufhin in einer zweiten Iteration in den Suchbegriff integriert. Dies ermöglicht eine detailliertere Suche und eine breitere Datenbasis. Dazu wird der

Suchbegriff wie in Tabelle 2.2 beschrieben angepasst.

UI (in title)	Strategie (in title)
UI	Monolithisch
UIs	Monolithic
User interface	Micro-frontend
User interfaces	Micro frontend
Frontend	Self-Contained-System
Frontends	

Tabelle 2.2: Suchprozess - Suchbegriffermittlung zweite Iteration

Die daraufhin gefundenen Publikationen sind erneut durch den Auswahlprozess in Abbildung 2.2 auf Seite 7 gelaufen. Die zweite Iteration ergab zwei weitere relevante Werke **No. 7** und **No. 8**.

Datenanalyse

Die vorher gesammelten Daten werden im nächsten Schritt aggregiert und in den Kontext zueinander gestellt. Das Ergebnis der Datenanalyse ist ein zusammenfassender Bericht mit allen Kernpunkten bezogen auf die vorher formulierten Forschungsfragen in Form einer tabellarischen Liste. Das Ziel ist es, aggregierte Daten über die Erkenntnisse der Literatur in Abhängigkeit der verschiedenen Integrationsverfahren **(RQ1)** von User Interfaces in Microservices zu erzeugen. Pro Strategie werden Vorteile **(RQ2)** und Nachteile **(RQ3)** aufgezeigt. Außerdem werden sowohl generelle Herausforderungen **(RQ4)** und Richtlinien **(RQ5)**, als auch bezogen auf die herausgearbeiteten Strategien ausgeführt.

2.3.2 Case Study

Um weitere, praktische Erkenntnisse zu den formulierten Forschungsfragen zu erhalten, wird im nächsten Schritt eine Case Study mit Experteninterviews durchgeführt. Die Struktur der Case Study hält sich an den von Runeson und Höst (2009) festgelegten Checklisten und Richtlinien. Im Folgenden wird der Prozess der Case Study beschrieben.

Case Study Design

Im Case Study Design wird der konzeptionelle Rahmen und Zielsetzungen der Case Study festgelegt. Im Vordergrund steht ein Erkenntnisgewinn durch Einblicke in Praktiken von Industriepartnern. Mithilfe der gewonnenen Erfahrungen sollen die formulierten Forschungsfragen aus einer praktischen Perspektive betrachtet werden.

Vorbereitung der Datensammlung

Als vorbereitende Maßnahmen für die Datensammlung wird ein Case Study Protokoll formuliert. Das Protokoll ist abgeleitet aus den zugrunde liegenden Forschungsfragen. Das Protokoll ist im Anhang C.1 zu finden. Für die eigentliche Datensammlung wird ein semi-strukturiertes Interview als Methode gewählt. Beim semi-strukturierten Interview werden dem Partner eine Mischung aus offenen und geschlossenen Fragen gestellt. Die Zielsetzung ist dabei sowohl exploratorischer als auch deskriptiver Art. (Runeson & Höst, 2009) Für das Interview wird ein Interview Guide entworfen, welcher konkrete Fragestellungen enthält. Die tatsächlich gestellten Fragen können je nach Situation und Detailgrad abweichen. Der Interview Guide ist in Anhang C.2 beigelegt.

Wie in den Richtlinien von Runeson und Höst (2009) beschrieben, ist das Interview in vier Phasen aufgeteilt.

1. Vorstellung des Forschungsziels und wie mit den gesammelten Daten verfahren wird
2. Einleitende, offene Fragen zum Projektkontext und Hintergrund
3. Detailierte, meist geschlossene Fragen, welche abgeleitet von den Forschungsfragen werden
4. Zusammenfassung der wesentlichen Ergebnisse um Missverständnisse zu vermeiden und Feedback zu erhalten

Convenience

Bei der Auswahl der Interviewpartner werden mehrere Kriterien in Betracht gezogen. Um Bias zu vermeiden, sollen möglichst Interviewpartner mit unterschiedlichen Sichtweisen ausgewählt werden. Dazu gehört, dass sich die Interviewteilnehmer möglichst in einem anderem Projektkontext bewegen und andere Rollen im Projekt einnehmen. Da qualitative Daten erhoben werden, wird die Anzahl der Teilnehmer auf zwei beschränkt.

Konkret werden zwei Teilnehmer eines deutschen Softwareherstellers und IT-Dienstleisters befragt, um einen Überblick über die in der Praxis eingesetzten Verfahren zu erhalten. Die Teilnehmer arbeiten in unterschiedlichen Rollen und Projekten und sind damit geeignet für das Interview. Die Partner haben in der Vergangenheit unterschiedliche Strategien für die Frontend-Integration in Microservice Anwendungen verwendet. Die Fragen werden dadurch basierend auf unterschiedlichen Erfahrungsschätzen beantwortet, was eine heterogene Sichtweise auf die Forschungsfragen ermöglicht.

Ausführung

Die Daten aus den Experteninterviews werden ausgewertet, in einer tabellarischen

Übersicht dargestellt und auf die Forschungsfragen abgebildet. Der Interviewleitfaden und das Transkript ist im Anhang der Arbeit zu finden.

Die Durchführung der Interviews wird aufgenommen. Die Aufnahmen werden zu Text transkribiert. Anschließend erhalten die Interviewpartner das Transkript zugestellt, um Missverständnisse zu vermeiden und Erweiterungen hinzuzufügen.

Datenanalyse

Bei der Datenanalyse wird zwischen quantitativen Daten und qualitativen Daten unterschieden. Bei den Interviews werden hauptsächlich qualitative Daten erhoben. Zusätzlich wird die Validität der Daten betrachtet und eingeschätzt. Unkonkrete Fragestellungen oder unterschiedliche Interpretationen der Frage können zu gering validen Daten führen.

Die Daten werden im Anschluss tabellarisch aufbereitet und ausgewertet. Eine detaillierte Auswertung der Daten befindet sich im Anhang C.3. Die Tabelle zeigt Aspekte zu den Forschungsfragen **RQ1** bis **RQ4**. Außerdem stellt sie die Korellation zwischen Case Study und Literaturanalyse dar.

Reporting

Im Reporting wird ein Review der Interviews durchgeführt. Dabei soll diskutiert werden, inwieweit die im Case Study Design formulierten Zielsetzungen erreicht wurden.

2.3.3 Aktionsforschung

Aktionsforschung verbindet Forschung und Praxis. Ziel der Aktionsforschung ist es, ein Problem aus der Praxis zu lösen. Dabei nimmt der Aktionsforscher bewusst Einfluss auf das Feld (Avison et al., 1999). Im Kontext der Forschungsarbeit, werden ausgewählte Strategien für Frontend-Integretation in Microservices umgesetzt.

Aktionsforschung ist eine iterative Forschungsmethode. Das Vorgehen der Aktionsforschung nach Susman und Evered (1978) gliedert sich in fünf Schritte, die periodisch wiederholt werden.

Die verschiedenen Phasen werden iterativ durchlaufen. Zu Beginn des Zyklus wird die Situation analyisiert und das weitere Vorgehen geplant. Am Ende des Zyklus werden die gefundenen Ergebnisse festgehalten und die nächste Iteration begonnen. Die Schritte erfolgen in Zusammenarbeit zwischen Forschung und Praxis. (Susman & Evered, 1978)

In der Arbeit wird die Aktionsforschung anhand der Anwendung JValue durchgeführt. Die Ergebnisse der Aktionsforschung sind im Kapitel 2.5 tabellarisch festgehalten.

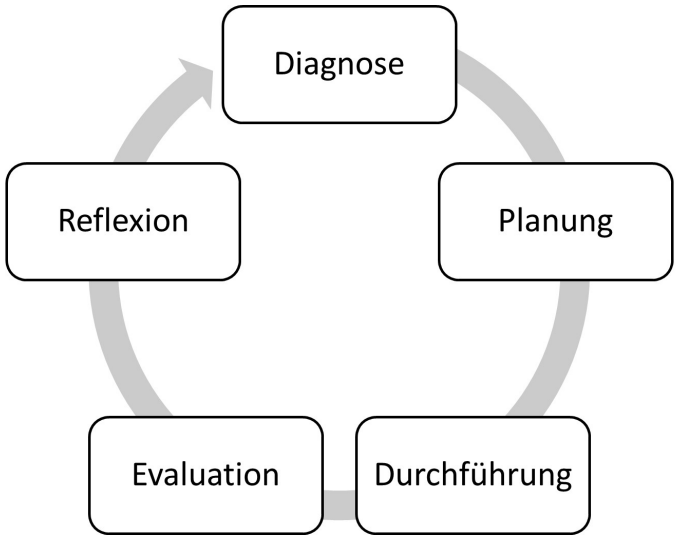

Abbildung 2.3: Zyklus der Aktionforschung (Susman & Evered, 1978)

2.4 Theorie

2.4.1 Überblick User Interface Strategien

Die Sektion stellt die Ergebnisse der Literaturanalyse und der Case Study gebündelt dar. Zuerst werden Aspekte zu **RQ1** dargelegt. Die nachfolgenden Forschungsfragen beziehen sich anschließend immer auf einen Aspekt von **RQ1**.

RQ1: Strategien

No.	Beschreibung	Quellen
S1	Monolithische UI	(Harms et al., 2017) (Li et al., 2020) (Tilak et al., 2020) (Yang, 2022) (Prajwal et al., 2021) (Peltonen et al., 2021)
S2	Micro Frontend	(Harms et al., 2017) (Tilak et al., 2020) (Prajwal et al., 2021) (Peltonen et al., 2021)
S3	Self-Contained-System	(Harms et al., 2017)

Tabelle 2.3: Strategien User Interfaces in Microservices (RQ1)

Bei einer monolithischen UI ist ein eigener, seperater Microservice für die gesamte

Benutzeroberfläche zuständig. Andere Microservices können Daten für das User Interface liefern, erzeugen selbst aber keine Komponenten am Frontend.

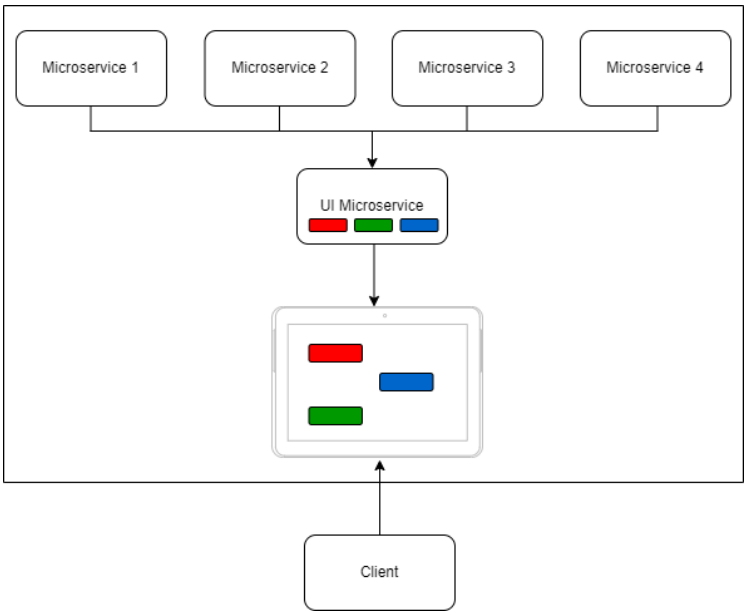

Abbildung 2.4: Monolithische UI

Abbildung 2.4 stellt den Aufbau einer Anwendung mit monolithischer UI vereinfacht dar. Hier steuert der UI Microservice alle Komponenten für das finale User Interface bei. Die Literaturanalyse zeigt auf, dass dieser Architekturstil in allen betrachteten Werken behandelt wird.

Die Micro Frontend Architektur sieht keinen eigenen Microservice vor, der eigenständig für das Frontend verantwortlich ist. Stattdessen tragen die Microservices zusätzlich zu ihrer Business Logik auch die Komponenten für das User Interface bei. In Abhängigkeit von der technischen Umsetzung gibt es einen Microservice, der die UI Komponenten der Microservices zum Endergebnis zusammensetzt.

Bei Micro Frontends handelt sich um einen Architekturstil, der auch in der Literatur erst vor wenigen Jahren an Bedeutung gewonnen hat. Ebenfalls zeigt die Analyse, dass nur wenige Artikel veröffentlicht sind, die auch die Probleme und Schwachstellen von Micro Frontends aufzeigen und diskutieren.

Self-Contained-System (SCS) ist ein breit gefächerter Begriff, welcher in Harms et al. (2017) auf Frontends in Microservices übertragen wird. SCS sind eine weitere Möglichkeit um User Interfaces in Microservices zu integrieren. Im Unterschied zur Micro Frontend Architektur, enthält jeder Service bei Self-Contained-Systems ein eigenes, separates User Interface. Der Begriff Self-Contained-System und Microservice wird in der Literatur oft gleichgesetzt. Dabei enthält ein SCS meist mehr Funktionalität als ein Microservice. SCS sind hauptsächlich autonome An-

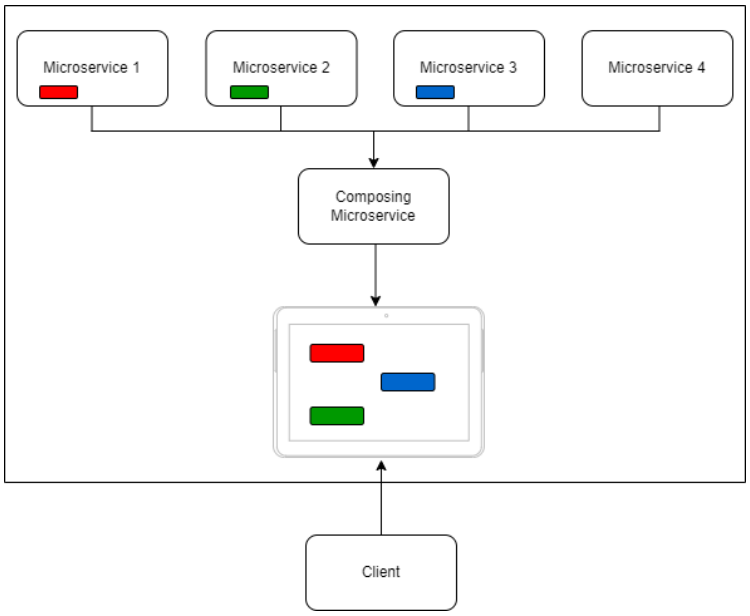

Abbildung 2.5: Micro Frontend

wendungen, die über Hyperlinks mit anderen SCS kommunizieren können. Der Client wird über einen Reverse Proxy an das von ihm geforderte User Interface weitergeleitet. (Harms et al., 2017)

Die Literaturanalyse zeigt, dass bislang kaum Artikel SCS im Zusammenhang mit Microservices und User Interfaces thematisieren.

2.4.2 Monolithische UI

RQ2: Vorteile

No.	Beschreibung	Quellen
$V1_{S1}$	Geringeres Risiko bei Neuentwicklungen	(Harms et al., 2017)
$V2_{S1}$	Bessere Unterstützung von Event-Driven Architektur	(Harms et al., 2017)
$V3_{S1}$	Bessere Möglichkeiten zum Debugging	(Peltonen et al., 2021)
I-$V4_{S1}$	Höhere Entwicklungseffizienz	Interview 2

Tabelle 2.4: Vorteile monolithisches User Interface (RQ2)

Das Risiko mit monolithischem Frontend bei Neuentwicklung von Anwendungen ist geringer als bei anderen Architekturen. In frühen Stadien der Entwicklung,

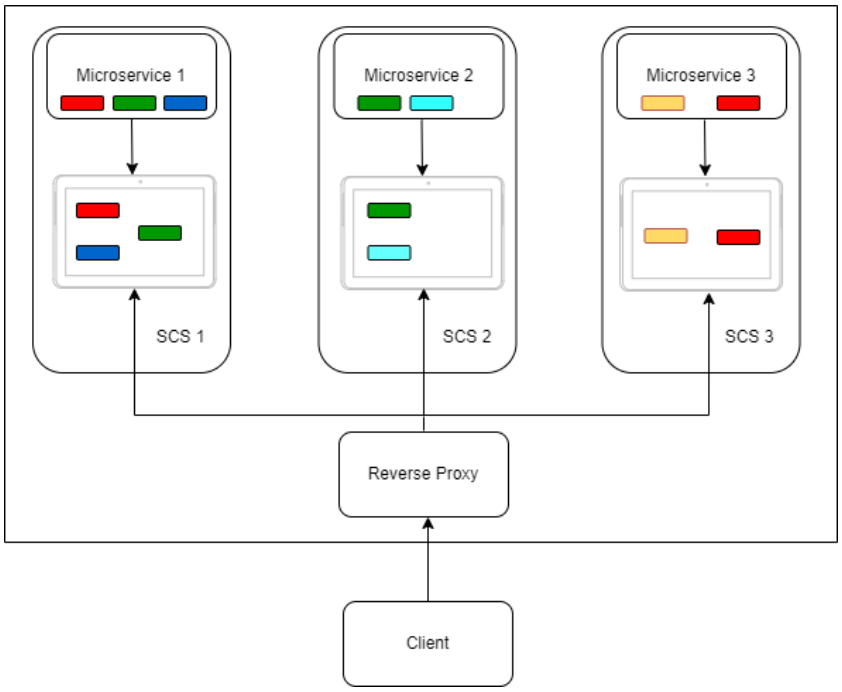

Abbildung 2.6: Self-Contained Systems

wenn der Business-Prozess der Anwendung noch nicht eindeutig ist, sind in der Regel Änderungen über Service-Grenzen hinweg nötig. Diese Änderungen sind bei monolitscher Frontend-Architektur mit geringerem Aufwand verbunden als bei den anderen Varianten. (Harms et al., 2017)

Der Einsatz einer Event-Driven Architektur ist bei monolitischer UI besser möglich. In diesem Fall wird ein Event-Bus zwischen UI Microservice und anderen Microservices eingesetzt. Dieser ist für die Koordinierung von Events in einer Anwendung verantwortlich. Bei anderen Strategien ist dieser Ansatz komplexer umzusetzen. (Harms et al., 2017)

Zuletzt ist Debugging in monolithischen UI's besser möglich. Damit verbunden ist eine effektivere Softwareentwicklung und eine implizite Einsparung von Kosten bei der Anwendungsentwicklung. (Peltonen et al., 2021)

I-V1$_{S1}$: Höhere Entwicklungseffizienz

Der Teilnehmer sieht in der monolithischen UI einen Vorteil in der Entwicklungseffizienz. Dieser wird dadurch erreicht, dass in dem Monolithen modular Komponenten in eigene Bibliotheken ausgelagert werden, welche zur Compilezeit wieder in die Haupt-Anwendung eingebunden werden. Dadurch wird mehr Entwicklungseffizienz geschaffen als durch die Verteilung in eigenständige Micro Frontends.

> „Das hat mehrere Gründe, weshalb wir uns für dieses monolithische Frontend entschieden haben. Zum Einen haben wir ein einziges Plattform Team, das heißt wir haben ein Team das an dem Code entwickelt. Der Overhead wäre allein schon organisatorisch mit mehreren Projekten viel zu groß. Wir haben dann entschieden, dass wir dann modular innerhalb dieses Monolithen entwickeln, also verschiedene Dinge in eigene Komponenten oder Libraries auslagern. Dann erreichen wir dadurch eigentlich mehr Entwicklungseffizienz, als wenn wir das künstlich verteilen und ständig immer in eigene Frontend Anwendungen wechseln müssen.“ (Interview 2)

RQ3: Nachteile

No.	Beschreibung	Quellen
$N1_{S1}$	Monolithische UI als Flaschenhals	(Harms et al., 2017) (Tilak et al., 2020)
$N2_{S1}$	Wartbarkeit und Koordination	(Tilak et al., 2020) Interview 1
$N3_{S1}$	Performance	(Harms et al., 2017)

Tabelle 2.5: Nachteile monolithisches User Interface (RQ3)

Monolithische UI's können sich als Flaschenhals einer Anwendung darstellen. Längere Releasezyklen und größere interne Kopplung beeinflussen Anwendungen mit monolithischem Framework bei der Evolutionsfähigkeit. Diese kann nicht in gleichem Ausmaß skalieren, wie eine Anwendung mit Modularisierung oder Micro Frontends. (Harms et al., 2017; Tilak et al., 2020)

Der Nachteil $N1_{S1}$ wirkt sich auch negativ auf die Wartbarkeit der Anwendung dar. Je größer die Anwendung wird, desto mehr Abhängigkeiten und Kopplung wird es bei einem monolithischem UI Framework geben. Wenn viele Teams an der Anwendung entwickeln führen Änderungen an dem User Interface zu größerem Koordinationsaufwand. Das bremst die Wartbarkeit der Anwendung aus. (Tilak et al., 2020)

Wie $V1_{S2}$ beschreibt, hat eine monolithische UI Nachteile im initialen Laden der Seite. Dadurch, dass initial bei monolithischer UI mehr Daten übertragen werden, ist die Latenz anfänglich größer. (Harms et al., 2017)

RQ4: Herausforderungen

Layering von monolithischen User Interfaces in microservice-basierten Systemen kann sich als Herausforderung darstellen. Die referenzierten Publikationen Yang (2022) und Li et al. (2020) stellen dabei Frameworks vor, die bei der Modulbilung einer Anwendung unterstützen sollen.

No.	Beschreibung	Quellen
$H1_{S1}$	Layering bei monolithischer Architektur	(Yang, 2022) (Li et al., 2020)
$H2_{S1}$	Technologie-Vielfalt	(Li et al., 2020)

Tabelle 2.6: Herausforderungen monolithisches User Interface (RQ4)

Auch bei monolithischen User Interfaces ist Layering möglich. Dabei geht das Paper auf die bekannten MV* und Model-View-Controller (MVC) Pattern ein. Darüber hinaus stellt es einen neuen Ansatz MVVC vor. Je nach Anforderungen der Anwendung haben die Pattern unterschiedliche Vor-/Nachteile. Bei der Auswahl des Frameworks in einer monolithischen Struktur, sollen die Anforderungen der Anwendung also auch berücksichtigt werden. Im Falle der Publikation wird auf die Anforderungen eines Background-Management Systems eingegangen. Dabei ist die initiale Latenz beim Start der Anwendung für den User möglichst gering zu halten. Zusätzlich werden viele Create, Read, Update, Delete (CRUD) und wenige Änderungen am Domain Object Model (DOM) vorkommen. Deshalb ist es bei der Beispielanwendung wichtig, einen hohen Datendurchsatz zu erzielen. Das vorgestellte MVVC Pattern zwingt den Entwickler dabei, den Schwerpunkt bei der Entwicklung auf die Logik innerhalb des Daten-Layers zu legen. Die Ausarbeitung der Anforderungen und die Auswahl des daraufhin geeigneten Patterns stellt Unternehmen bei der Entwicklung der Anwendung vor eine Herausforderung. (Yang, 2022)

Ebenso wie das Design-Pattern, stellt auch die Repository-Struktur eine Herausforderung bei monolithischen Frontends dar. Li et al. (2020) stellt dafür ein weiteres Frontend-Framework vor, welches ähnlich dem MVC ist. Damit soll durch Wiederverwendung von Komponenten eine geringere Kopplung und schnellere Entwicklung möglich sein.

Die referenzierten Publikationen zu Herausforderung $H1_{S1}$ beziehen sich auf monolithische Frontends. Die Herausforderung kann aber auch generell und unabhängig von der Architektur gelten.

2.4.3 Micro Frontend

RQ2: Vorteile

Beim Aspekt Performance gibt es Unterschiedliche Eindrücke in der Literatur. Harms et al. (2017), Peltonen et al. (2021) und Prajwal et al. (2021) zeigen auf, dass Micro Frontends die beste Performance im initialen Laden der UI unter Einsatz von Caching-Systemen. Gerade wenn mehrere Web-Frameworks eingesetzt werden, steigt andererseits dabei auch das Übertragungsvolumen von Daten an

No.	Beschreibung	Quellen
$V1_{S2}$	Performance im initialen Laden (mit Caching)	(Harms et al., 2017) (Peltonen et al., 2021) (Prajwal et al., 2021) Interview 1
$V2_{S2}$	Anpassung an Organisationsstrukturen der Entwicklungsteams	(Harms et al., 2017) (Tilak et al., 2020) (Peltonen et al., 2021)
$V3_{S2}$	Unabhängige Technologie-Stacks in den Entwicklungsteams	(Tilak et al., 2020) (Peltonen et al., 2021) (Prajwal et al., 2021)
$V4_{S2}$	Unabhängige Entwicklung und Deployment	(Peltonen et al., 2021) (Prajwal et al., 2021)
$V5_{S2}$	Bessere Testbarkeit	(Peltonen et al., 2021) (Prajwal et al., 2021)
$V6_{S2}$	Bessere Fehlertoleranz	(Peltonen et al., 2021) (Prajwal et al., 2021)

Tabelle 2.7: Vorteile Micro Frontend (RQ2)

den End-Nutzer. Der Grund liegt hierbei an der Runtime Integration der UI Komponenten. Diese werden bei Micro Frontends in der Praxis normalerweise zur Laufzeit geladen und nicht zur Build-Zeit. Dies kann das Kundenerlebnis beeinträchtigen.

Ein Vorteil, der in den meisten Publikationen erwähnt wird, ist die bessere Eingliederung der Anwendung in mehrere Entwicklungsteams. Jedes Team verantwortet einen Service inklusive dem Teil des User Interfaces, das den Service repräsentiert. Bei monolithischen Frontend existiert in der Regel ein seperates Entwicklungsteam, welches für das gesamte User Interface zuständig ist. Dies ist bei Micro Frontends nicht mehr notwendig. Stattdessen können die Teams autonom und unabhängig Änderungen an der UI durchführen. Damit kann Koordinationsaufwand und Kommunikation zwischen den Teams eingespart werden. (Harms et al., 2017; Peltonen et al., 2021; Tilak et al., 2020)

Des Weiteren können die Entwicklungsteams den Technologie-Stack frei wählen. Die Auswahl des Frontend-Frameworks ist nicht mehr entscheidend, da die UI Komponenten zur Laufzeit zusammengefügt werden. Entscheidet sich ein Team beispielsweise für React als Web-Framework, müssen andere Teams nicht das gleiche Framework verwenden, sondern können autonom entscheiden, welches Web-Framework eingesetzt werden soll. Damit können Kompetenzen und Know-how besser berücksichtigt werden. (Peltonen et al., 2021; Prajwal et al., 2021; Tilak et al., 2020)

Ein weiterer Vorteil von Micro Frontends, der mit Vorteil $V2_{S2}$ zusammenhängt, ist die Möglichkeit, Software unabhängig zu entwickeln und zu betreiben. Eine Änderung an einem Micro Frontend beeinflusst andere Services nicht. Dadurch kann bei großen Anwendungen jedes Team unabhängig und parallel entwickeln. Auch das Deployment wird damit erleichtert, da es ausreicht, das eigene Micro Frontend neu zu erzeugen. Es muss nicht bei jeder Änderung die gesamte Anwendung neu erzeugt werden. Dies kann zu schnelleren Test- und Releasezyklen führen und hilft bei Continuous Integration und Continuous Delivery. (Peltonen et al., 2021; Prajwal et al., 2021)

Der Aspekt Testbarkeit wird in der Literatur unterschiedlich betrachtet. Peltonen et al. (2021) und Prajwal et al. (2021) sind sich einig, dass es bei Micro Frontends einfacher ist Teilaspekte des Systems zu testen. Der Entwickler muss bei einer Änderung nicht die gesamte Test-Suite erneut ausführen. Prajwal et al. (2021) stellt aber zusätzlich dar, dass es aber aufgrund von erhöhter Komplexität schwieriger ist, das Gesamtsystem zu testen.

Zuletzt ermöglichen Micro Frontends eine bessere Fehlertoleranz gegenüber anderen Strategien. Im Falle eines Ausfalls von einem Service, muss nicht die gesamte Anwendung heruntergefahren werden. Stattdessen kann die Laufzeit einen Fehler in einem Micro Frontend erkennen und dem Nutzer einen entsprechenden Hinweis geben. Dies ist ein großer Vorteil gegenüber monolithischen UI's. (Peltonen et al., 2021; Prajwal et al., 2021)

RQ3: Nachteile

Bei Micro Frontends können Probleme auftreten, wenn das System unter hoher Last ist. In der Publikation zeigen Harms et al. (2017) auf, dass eine Micro Frontends Anwendung bei hoher Last unter Umständen fehlerhafte Daten liefern kann. In dem in der Studie gezeigten Test, werden kumulierte Artikelpreise in einem Warenkorb analysiert. Die Preise werden von unterschiedlichen Microservices übermittelt. Der Test zeigt, dass der im Frontend angezeigte Gesamtpreis in einigen Fällen fehlerhaft ist.

Außerdem erfordern Änderungen in Querschnittsthemen bei Micro Frontends einen hohen Koordinationsaufwand. Gerade bei neueren Anwendungen kann es vorkommen, dass eine Änderung im Backend mehrere Microservices betrifft. Das kann zum Beispiel der Fall sein, wenn sich ein Business Prozess ändert, oder bei der ersten Implementierung nicht vollständig oder richtig bedacht wurde. Bei Micro Frontends steuern die Entwicklungsteams einen eigenen Teil zum Frontend bei. Die korrespondierende, übergreifende Änderung im Frontend muss nun von mehreren Teams koordiniert werden. Dieser Mehraufwand an Koordination bei Änderungen über Service Grenzen hinweg, wird in den meisten Publikationen als Nachteil von Micro Frontends dargestellt. (Harms et al., 2017; Peltonen et al., 2021; Prajwal et al., 2021)

No.	Beschreibung	Quellen
$N1_{S2}$	Interoperabilität unter hoher Last	(Harms et al., 2017)
$N2_{S2}$	Koordination bei Änderungen in Querschnittsthemen	(Harms et al., 2017) (Peltonen et al., 2021) (Prajwal et al., 2021)
$N3_{S2}$	Komplexere Struktur	(Tilak et al., 2020) (Peltonen et al., 2021) Interview 1
$N4_{S2}$	Code- und Dependency Redundanzen	(Peltonen et al., 2021) (Prajwal et al., 2021)
$N5_{S2}$	Risiko von Code-Divergenz	(Prajwal et al., 2021)
$N6_{S2}$	Eingeschränkte Monitoring Möglichkeiten	(Peltonen et al., 2021)
$I\text{-}N7_{S2}$	Schwierigere Fehlerbehandlung	Interview 1
$I\text{-}N8_{S2}$	Aufwendigere Integration bestimmter Wartbarkeitsszenarien	Interview 1
$I\text{-}N9_{S2}$	Höhere Netzwerklast	Interview 1

Tabelle 2.8: Nachteile Micro Frontend (RQ3)

Ein Anwendung auf Micro Frontends herunterzubrechen, kann eine große Herausforderung sein. Durch den Einsatz von Micro Frontends steigt die Komplexität einer Anwendung auf technischer und organisatorischer Ebene. Das zeigt sich gerade dann, wenn eine Anwendung mehrere Clienttypen unterstützen soll. Falls verschiedene Web-Technologien eingesetzt werden, wird es deutlich komplexer die Anwendung auf unterschiedliche Endgeräte auszurichten. Auch die Integration von mehreren Child-Projekten erfordert einen höheren Aufwand bei der Zusammenführung des Frontends. Durch die verteilte Struktur steigt die Menge an Dateien und damit die Komplexität des Projekts. (Peltonen et al., 2021; Tilak et al., 2020)

Ein weiterer Nachteil beim Einsatz von Micro Frontends sind Redundanzen und Duplikationen. Durch unabhängige Entwicklung der Micro Frontends in den Entwicklungsteams, besteht das Risiko von Code- und Dependency Redundanzen. Gemeinsam genutzte Librarys werden von den Services mehrfach verwendet und geladen. Wenn mehrere Micro Frontends beispielsweise React als Web-Technologie einbinden, wird die Library bei jedem Frontend geladen und zur Laufzeit an das User Interface übertragen. Damit erhöht sich implizit die Datenmenge die zum Enduser übertragen wird und die Performance wird beeinträchtigt. Außerdem kann ohne Kommunikation zu anderen Entwicklungsteams Quellcode-Duplikation passieren. Dies führt zu Parallelarbeit und erhöhtem Ressourceneinsatz. Damit begünstigen Micro Frontend Architekturen sogenannte Islands of Knowledges in

einem Unternehmen. Die Gesamtkosten und Zeit bei der Entwicklung der Anwendung können dadurch beeinträchtigt werden. (Peltonen et al., 2021; Prajwal et al., 2021)

Wenn ein bestehendes microservice-basiertes System auf Micro Frontends angepasst wird, besteht das Risiko von Code-Divergenz. Durch die Entwicklung und Anpassung der Organisation, können Abweichungen zum Altsystem entstehen. (Prajwal et al., 2021)

Die Entwicklung und der Betrieb von Micro Frontends kann sehr komplex und langwierig sein. Bei der Entwicklung müssen Unternehmen beachten, dass das Testen und Entwickeln von Interaktionen zwischen den Micro Frontends sehr komplex sein kann. Außerdem stellt sich dar, dass Monitoring und Tracking der Anwendung schwieriger wird. (Peltonen et al., 2021)

I-N7$_{S2}$: Schwierigere Fehlerbehandlung

In der Literatur wird die bessere Fehlertoleranz durch Micro Frontends hervorgehoben. Diese erfordert aber ein deutlich umfangreicheres Fehler-Handling, was die Komplexität der Anwendung erhöht. Wenn einzelne Micro Frontends ausfallen, müssen diese Fehler in der Container-Anwendung abgefangen werden, da sie sonst an den User weitergeleitet werden.

> „Also als erstes muss man sich die die Wartbarkeit Aspekte mit Organisation anschauen. Eigentlich sagt man ja heutzutage „Inverse Convey“, das heißt die Organisation richtet sich nach der Architektur. Beim Frontend ist das nicht immer so einfach. Wenn man sehr viele Teams hat, die parallel crossfunktional entwickeln können, dann spricht meiner Meinung nach nichts gegen eine Micro Frontend Architektur mit den Trade-Offs die man sich dann einkauft. Schwierigere Konsistenz, Fehler Handling, Verfügbarkeit von einzelnen Funktionen. Wenn das handhabbar ist und von der Priorität her weniger wichtig ist als die Entwicklungseffizienz und die Autonomie der einzelnen Teams, dann gewinnt eben das.“ (Interview 2)

I-N8$_{S2}$: Aufwendigere Integration bestimmter Wartbarkeitsszenarien

Die Literatur führt unter dem Nachteil N2$_{S1}$ die Probleme der monolithischen UI unter dem Aspekt der Wartbarkeit auf. Im Interview stellt sich heraus, dass häufig ein bestimmtes Wartungsszenario bei der Entwicklung der Anwendung vorkommt. Dies wirkt sich dahingehend aus, dass mehrere Services betroffen sein werden. Das Wartungsszenario wurde in der Architekturentscheidung der Anwendung priorisiert. Die Änderungen müssen dabei schnell ausgeliefert werden.

> „Fast in jedem Software-System das gesetzlich getrieben ist, müssen einige Änderungen sehr schnell aufgrund von Gesetzesänderungen eingebaut werden. Das trifft vor allem auf Banken und Versicherungen

> zu. Und wenn es das Unternehmen nicht schafft zu diesem Zeitpunkt die Änderung einzubauen, dann gibt es ein Problem, weil das System nicht mehr korrekt arbeitet. Bei uns ist das historisch auch immer schon extrem wichtig gewesen. Aber das ist das worauf das Back-End optimiert ist, und das ist eben nicht die User Journey. Das ist das Problem, deshalb vertragen sich das Backend und das Frontend nicht besonders. Der Schnitt wurde also darauf optimiert, dass wir, wenn eine Gesetzesänderung kommt, diese möglichst schnell einbauen können. Das ist dem Kunden am allerwichtigsten. Und es ist eben nicht so wichtig, dass er sich konsistent durch ein System bewegen kann, dass sich ideal mit seiner User Journey verhält. Und wenn diese Entscheidung getroffen wird, dann gibt es bei der Frontend Integration sofort einige Probleme.“ (Interview 1)

Eine Micro Frontend Architektur würde den Einbau der Änderungen deutlich erschweren. Der Aspekt hängt mit $N2_{S2}$ zusammen, der in der Literatur bereits häufig genannt wurde. Wenn es die Anwendungsentwicklung erfordert, schnell und effizient bestimmte Einflüsse von außen in die Anwendung zu integrieren, dann ist eine Micro Frontend Architektur unter Umständen nicht geeignet.

I-N9$_{S2}$: Höhere Netzwerklast

Ein weiterer Nachteil, der in den Interviews aufgeführt wurde, ist die erhöhte Netzwerklast. Im Kontext der Anwendung werden die einzelnen Seiten oft mit statischen Daten vorbelegt, die aus dem Backend geholt werden. Der Zustand muss jeweils an die weiteren Micro Frontends verteilt werden. Dies erfordert mehr Requests an das Backend um die gemeinsam genutzten Daten zu verteilen und damit einen erhöhten Datentransfer.

> „Vor allem im Vergleich zu den vorherigen Micro Frontends haben wie mit dem monolithischen Frontend einen Vorteil im Bereich Performance. Das ist auch einer der Hauptgründe, warum man zurück zum monolithischen Frontend ist, weil der Micro Frontend Wechsel Zeit benötigt. Und das aus zwei Gründen, einmal natürlich, weil ein neues Frontend im Browser geladen wird. Aber der andere Grund auch, weil sich das Frontend beim Start relativ stark mit statischen Daten aus dem Backend vorbelegt. Es lädt aus dem Backend verschiedene Default-Werte und konfiguriert sie. Dann ist ein Frontend zu anderem Micro Frontend Wechsel natürlich besonders teuer, weil dieser Zustand übertragen werden muss. Alternativ müsste sich das neue Frontend vollständig neu initialisieren. Dadurch muss dieses sehr viele Netzwerkaufrufe machen um seine Werte zu erlangen.“ (Interview 1)

Das führt zu weiteren Folgeproblemen, unter anderem das Rate Limiting eines

Netzwerkadapters, um Denial-of-Service Angriffe zu vermeiden.

Die Nachteile $N1_{S2}$ bis $I\text{-}N9_{S2}$ zeigen, dass Micro Frontends nicht das Universalmittel für Frontend-Webtechnologien sind. Unternehmen müssen sich der Risiken bei Entwicklung und Betrieb bewusst sein. Der erhöhte Koordinationsbedarf kann die Entwicklung verlangsamen. Die vergrößerte übermittelte Datenmenge kann zusätzlich Performanceprobleme bei Endusern verursachen.

RQ4: Herausforderungen

No.	Beschreibung	Quellen
$H1_{S2}$	UX-Experience des Gesamtsystems	(Tilak et al., 2020) (Peltonen et al., 2021) Interview 2
$H2_{S2}$	Dokumentation	(Tilak et al., 2020)
$H3_{S2}$	Debugging	(Peltonen et al., 2021)
$H4_{S2}$	Technische Integration	(Harms et al., 2017) Interview 2

Tabelle 2.9: Herausforderungen Micro Frontend (RQ4)

Die Integration von Micro Frontends kann Unternehmen vor eine große Herausforderung stellen. Die in der Literatur häufigste aufgezeigte Herausforderung ist es, die User Experience (UX) des Gesamtsystems einheitlich und für den Endkunden ansprechend zu gestalten.

Die UX beschreibt das subjektive Nutzererlebnis eines bestimmten Produktes oder eines bestimmten Produktes. Sie beeinflusst die Qualität der Interaktion mit dem Endkunden und damit indirekt den Erfolg des Produktes. (Hassenzahl & Tractinsky, 2006)

Micro Frontends müssen als einzelne Komponenten entwickelt werden. Der Verantwortungsbereich bei der Umsetzung liegt bei den Teams. Jedoch sieht der Endnutzer das User Interface des Gesamtsystems. Durch die Aufteilung der Verantwortlichkeit für die UI in die verschiedenen Entwicklungsteams ist es schwierig, ein gutes Gesamtbild der Anwendung zu erreichen. Damit der Endnutzer eine gute UX erlebt, müssen sich die Komponenten zu einer konsistenten Oberfläche ergänzen. Zum Beispiel müssen Design Guidelines und Standards abgestimmt werden. Eine mögliche Lösung wäre ein einheitliches, gemeinsam genutzes Style-Sheet. Dieses wiederspricht den Prinzipien von Microservices und würde bedeuten, dass alle Micro Frontends von einer Ressource abhängen. Um ein einheitliches, ansprechendes Design zu erreichen, ist viel Kommunikation und Koordination zwischen den Entwicklungsteams notwendig. (Peltonen et al., 2021; Tilak et al., 2020)

Des Weiteren müssen Entwickler mehr Aufwand in Dokumentation stecken. Ein

Ziel von Microservices und Micro Frontends ist die Komponentarisierung und Modularisierung. Dadurch wird die Wiederverwendung ermöglicht. Dies impliziert, dass mehrere Entwicklungsteams die Möglichkeit haben, auf die Frontends zuzugreifen. Damit das fehlerfrei funktioniert ist es nötig, die Module ausführlich zu dokumentieren. (Tilak et al., 2020)

Zudem kann die Entwicklung von Micro Frontends sehr komplex und langwierig sein. Wie Nachteil $N2_{S2}$ zeigt, gibt es nur wenige Tools für das Tracking und Monitoring der Anwendung. Dadurch können Fehler nur spät und schwer entdeckt werden. Außerdem sind die Debugging Möglichkeiten eingeschränkt. (Peltonen et al., 2021)

Für den Erfolg der Anwendung entscheidend ist die technische Integration des Frontends. Um Micro Frontends oder SCS zu integrieren existieren die Softwaremuster API Gateway und Backend for Frontends (BFF).

API Gateway:

Eine direkte Kommunikation zwischen Frontend und Backend-Microservices ist in den meisten Anwendungen aus vielerlei Gründen nicht geeignet. In der Praxis werden in der Regel die Softwaremuster API Gateway und BFF verwendet. Ein API Gateway agiert als Brücke zwischen Frontend und Microservices. Die Aufrufe der Frontends werden durch das API Gateway geschlüsselt und an die Microservices weitergeleitet. Dabei verwenden alle Clients das gleiche API Gateway. Abbildung 2.7 zeigt den Aufbau eines API-Gateways Softwaremuster exemplarisch. Dabei ist es wichtig, dass das Gateway nicht für die Komposition des User Interfaces zuständig ist, sondern das Backend von Client Zugriffen abschottet.

Das API Gateway übernimmt dabei mehrere Funktionen. Zum Einen leitet es Aufrufe der externen Clients an die Microservices weiter. Des Weiteren verwaltet das Gateway auch User Authentifizierung. Bei direkter Kommunikation müsste bei jedem Microservice ein Modul für User Authentifizierung existieren. Dies ist durch den Einsatz des API Gateways nicht mehr nötig. Der Aspekt Security wird durch das API Gateway ebenfalls hervorgehoben. Es gibt nach außen hin nur noch eine Schnittstelle und damit weniger Angriffsmöglichkeiten auf die Downstream Services. (Montesi & Weber, 2016)

Ein API Gateway besitzt allerdings auch einige Nachteile. Zum Einen ist durch dieses Softwaremuster ein Single Entry Point und damit ein Single Point of Failure entstanden. Wenn das API Gateway ausfällt, kann die Anwendung nicht mehr verwendet werden, obwohl die Microservices noch funktionsfähig wären. Außerdem bildet sich dadurch eine monolithische Struktur des Gesamtsystems.

Bei API Gateways verwenden jegliche Arten von Clients das gleiche Gateway. Bei größeren Umgebungen führt eine mobile App unter Umständen unterschiedliche Aufrufe aus als die Web Anwendung. Es kann außerdem vorkommen, dass die

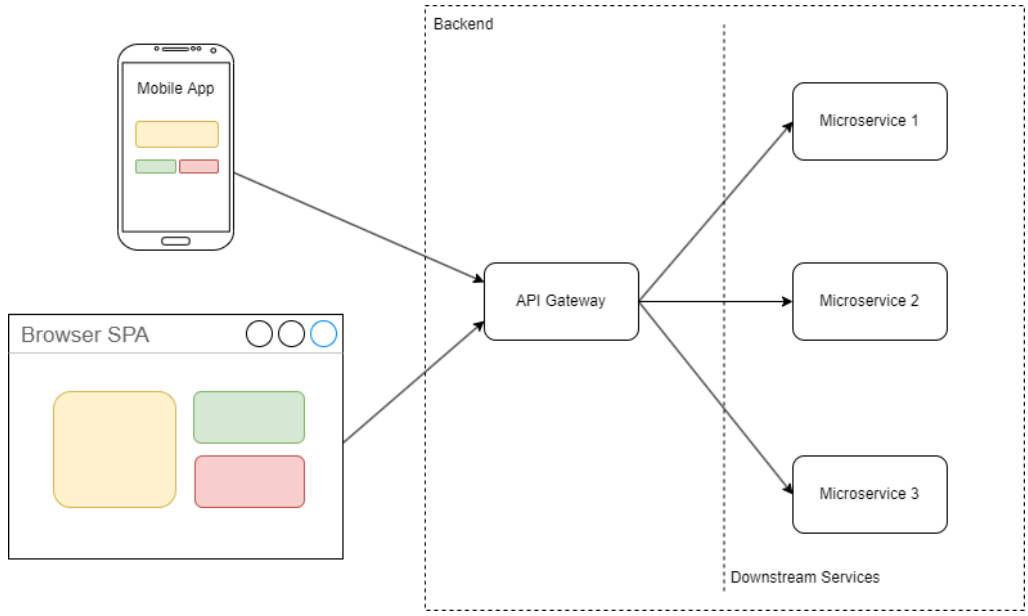

Abbildung 2.7: API Gateway Softwaremuster zur Kommunikation zwischen Frontend und Backend Services
Quelle: In Anlehnung an Newman (2015)

mobile Anwendung ein anderes User Interface und damit andere Daten für den User lädt. Das bedeutet für das API Gateway eine zusätzliche Komplexität. Zudem wird jeglicher Netzwerktransfer über das API Gateway geleitet. Bei vielen Nutzern und gleichzeitigen Aufrufen kann die Komponente unter einer enormen Last stehen. (Newman, 2015)

Die zusätzliche Komplexität im API Gateway führt oft dazu, dass ein eigenes Entwicklungsteam die Verantwortung über die Komponente besitzt. Dies stellt Unternehmen bei der Entwicklung vor eine Herausforderung, da für die verschiedenen Frontend Teams ein erhöhter Koordinationsaufwand entsteht. Auf der anderen Seite muss das API Gateway Team die Anforderungen aus den verschiedenen Frontend Teams steuern und priorisieren. Dies kann die Weiterentwicklung und Wartung der Anwendung verlangsamen.

Aus diesen Gründen hat sich mit dem BFF ein weiteres Softwaremuster etabliert, um Zugriffe aus dem Frontend an die Microservices weiterzuleiten.

Backends For Frontends:

Der wesentliche Unterschied zwischen BFF und API Gateway ist die Einführung weiterer Komponenten zur Unterstützung unterschiedlicher Clients. So gibt es für mobile Anwendungen ein eigenes BFF, welches speziell die Aufrufe aus mobilen Apps verwaltet. Der Zugriff aus Web Anwendungen wird über ein weiteres BFF geleitet.

Ein BFF verwaltet die gleichen Funktionalitäten wie das API Gateway. Es ist ebenfalls nicht Aufgabe des BFF, Frontend-Komponenten aus den Microservices zu aggregieren. Abbildung 2.4.3 stellt eine Microservice Umgebung unter Einsatz eines BFF dar.

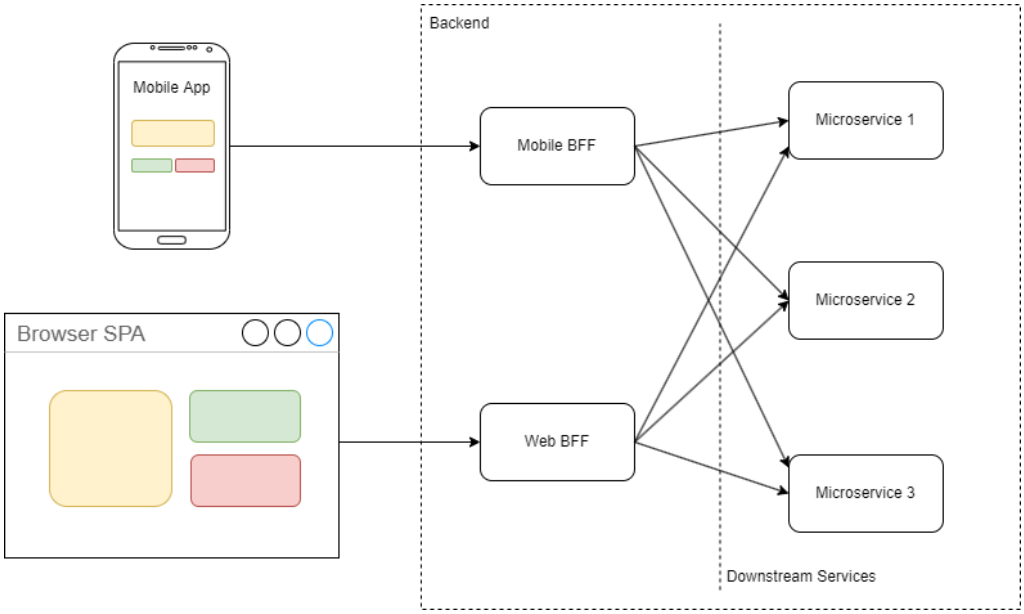

Abbildung 2.8: Backends for Frontends
Quelle: In Anlehnung an Newman (2015)

BFF bieten mehrere Vorteile gegenüber API Gateways. Die Unterstützung seperater Client-Typen reduziert die Komplexität der Codebases einzelner BFF's. Außerdem ermöglicht es den Teams eine größere Autonomität. Die Frontend Teams können ihre BFF's eigenständig verantworten. Dies kann sich positiv auf die Entwicklungsdauer auswirken. Durch die Einführung weiterer Komponenten verringert sich der Single Point of Failure. Wenn das Mobile BFF ausfällt, kann die Web Anwendung weiterhin lauffähig sein. (Newman, 2015)

Nachteilig wirkt sich die zusätzliche Komponente auf die Komplexität des Gesamtsystems aus. Dazu kommen Redundanzen dazu, da es sein kann, dass die mobile Anwendung die gleichen Aufrufe wie die Web Anwendung durchführt. (Newman, 2015)

Die Entscheidung hängt vor allem von den Anforderungen der Anwendung und der Organisation der Entwicklungsteams ab. $H4_{S2}$ trifft auf monolithische UI's, Micro Frontends als auch auf SCS zu und ist deshalb eine generelle Herausforderung bei der Integration von Frontends in Microservice Umgebungen.(Harms et al., 2017)

2.4.4 Self-Contained-Systems

RQ2: Vorteile

No.	Beschreibung	Quellen
$V1_{S3}$	Lose Kopplung im Frontend impliziert hohe Verfügbarkeit	(Harms et al., 2017)
$V2_{S3}$	Modifizierbarkeit	(Harms et al., 2017)
$V3_{S3}$	Anpassung an Organisationsstrukturen der Entwicklungsteams	(Harms et al., 2017)

Tabelle 2.10: Vorteile Self-Contained-Systems (RQ2)

Die Architektur von Self-Contained-Systems sieht eine lose Kopplung der Microservices vor. Damit bestehen kaum Abhängigkeiten zwischen den Frontends, vielmehr agieren diese als eigenständige Anwendungen. Dies bedeutet, dass im Fehlerfall eines SCS das Gesamtsystem weiter laufen kann. Das fehlerhafte SCS kann heruntergefahren werden und der Fehler kann unabhängig vom produktiven Betrieb der Anwendung behoben werden. Auf den Fehler kann der Nutzer während der Ausfallzeit mit einem Hinweis informiert werden. (Harms et al., 2017)

Des Weiteren ist die Modifizierbarkeit von Self-Contained-Systems besser als bei den anderen Strategien. Harms et al. (2017) vergleichen die notwendige Anzahl an zu ändernden Dateien für den Commit einer Änderung. Diese ist bei SCS am geringsten und lässt darauf schließen, dass SCS weniger komplex strukturiert ist.

Ebenfalls bieten Self-Contained-Systems den identischen Vorteil $V2_{S2}$. Durch die Aufteilung der Anwendung in kleinere eigenständige SCS, können sich die Entwicklungsteams entlang des Service-Schnitts ausrichten. Genauso wie bei der Micro Frontends Strategie können durch den Vorteil Koordinationsaufwand und Kommunikationsaufwand eingespart werden. (Harms et al., 2017)

Zusammenfassend lässt sich eine Häufung von Vorteilen bei Micro Frontend und monolithischer UI feststellen.

(RQ3 - RQ4): Nachteile und Herausforderungen

Aus der Literatur konnten keine Erkenntnisse über Nachteile als auch über Herausforderungen von Frontends in Self-Contained-Systems erhoben werden. Dies kann durch die kleinere Literaturbasis begründet werden.

2.4.5 Generelle Guidelines

Harms et al. (2017) stellen Guidelines für Frontend Integration in Microservices vor und brechen diese auf nicht-funktionale Anforderungen herunter. Für die Eva-

No.	Beschreibung	Quellen
G1	Nicht-funktionale Anforderungen	(Harms et al., 2017)
G2	Registry	(Tilak et al., 2020)
G3	Micro Frontends und Backend-Schnitt	Interview 1 Interview 2
G4	Planung des UX-Designs vor Frontend Integration	Interview 1
G5	Micro Frontends um Komponenten außerhalb der Domäne einzubinden	Interview 2

Tabelle 2.11: Guidelines (RQ5)

luation wird ein experimenteller Ansatz basierend auf verschiedenen Prototypen gewählt. Soll für die Anwendung Testbarkeit und Modifizierbarkeit wichtig sein, dann zeigen die Ergebnisse, dass SCS den größten Erfolg bringt. Die Micro Frontend Variante erreicht die besten Ergebnisse bei der Anforderung Performance.

Jedoch ist es wichtig, dass sich Performance im Kontext der Publikation auf das initiale Laden der Anwendung bezieht. Wie $V1_{S2}$ darstellt, kann sich der Micro Frontend Ansatz gerade bei größeren Anwendungen negativ auf die Performance auswirken, da mehr redundante Daten zur Laufzeit zum Client übermittelt werden müssen. Mit guten Ergebnissen beim initialen Laden der Anwendung einher geht also auch eine potentielle Verschlechterung der Performance im laufenden Betrieb unter Verwendung des Anwenders. (Peltonen et al., 2021)

Monolithische Frontends eignen sich weiterhin gut, wenn häufig Änderungen am Frontend über Service-Grenzen hinweg stattfinden. Da sich das User Interface in einem dedizierten Service befindet, muss mit keinem zusätzlichen Aufwand in Koordination und Kommunikation zwischen den Teams geplant werden. (Harms et al., 2017)

Ein weitere Möglichkeit ist der Einsatz einer internen Komponenten Registry. Entwickler können dort Microservices und Micro Frontends registrieren und für andere Teams zugreifbar machen. Die Plattform soll Teams bei der Entwicklung und Wartung von Microservices und Micro Frontends unterstützen. (Tilak et al., 2020)

Guideline G3 beschreibt die Ausrichtung der Micro Frontends auf den Backend-Schnitt der Microservices. Die im Interview dargestellte Erkenntnis bezieht sich auf die Wartbarkeit des Gesamtsystems. Diese verbessert sich bei Micro Frontends lediglich, wenn der Micro Frontend Schnitt auch auf den Backend-Schnitt passt. Wenn die Schnitte nicht aufeinander ausgerichtet sind, werden zusätzliche Service-Requests und von den Backend-Services gemeinsam genutzte Daten

problematisch. Unter diesen Umständen wird eine Migration sehr schwer oder unmöglich.

Guideline G4 stellt dar, dass bei einer Neuentwicklung der Anwendung das UX-Design vor der Micro Frontend Integration geplant werden sollte. Um Micro Frontends bestmöglich zu integrieren, bedarf es einer Ausrichtung der Micro Frontends anhand der User Journey. Zusätzlich ist es hilfreich ein gemeinsames Verständnis zwischen UX-Designern und Frontend-Entwicklern aufzubauen, da sich die UX entlang der technischen Grenzen orientieren muss.

Zuletzt kann es sinnvoll sein, Micro Frontends zu verwenden um optionale Features außerhalb der Domäne einzubinden (G5). Diese können von außen in die Anwendung integriert werden und zum Beispiel von Dritt-Parteien verantwortet werden. Der Vorteil darin ist es, dass es bei den eingebundenen Komponenten wenig Abhängigkeiten zur Haupt-Anwendung und es damit weniger Kopplung gibt. Das macht einen Einsatz von Micro Frontends sinnvoll.

Zusammenfassend stellt sich dar, dass die Case Study ähnliche Ergebnisse wie die strukturierte Literaturanalyse aufweist. Micro Frontends können nicht in jedem Kontext sinnvoll genutzt werden. Der in der Literatur als Vorteil ausgeführte Punkt Wartbarkeit, erweist sich jedoch gerade bei größeren Anwendungen in der Praxis als Nachteil. Wenn der Backend-Schnitt aufgrund anderer Prioritäten optimiert wird (I-N8$_{S2}$), dann ist eine Migration hin zu Micro Frontends sehr aufwendig oder unmöglich. Deshalb hat die Migration der Anwendung in der Case Study nicht funktioniert und die Architektur wieder zu einem monolithischen User Interface zurück geändert.

Im Interview wird das SCS als Zielzustand definiert. Das ist jedoch abhängig davon, dass sich der Backend-Schnitt entlang der User Journey orientiert und ist deshalb nicht immer möglich zu erreichen. Zusätzlich gibt es nur wenig wissenschaftliche Literatur über SCS und Frontend Integration.

2.5 Evaluation

Das Kapitel stellt den Kontext und die Durchführung der Aktionsforschung dar. Die Ergebnisse der Aktionsforschung dienen als Evaluation der aus den aus der Theorie gewonnenen Erkenntnisse. Anschließend wird daraus eine technische Guideline abgeleitet, welche die Migration einer Komponente hin zu einem Micro Frontend vereinfachen soll.

2.5.1 Kontext der Aktionsforschung

Die nachfolgende Sektion erläutert das Vorgehen der Aktionsforschung anhand der Anwendung JValue. Die Literatur und die Case Study thematisieren gleicher-

maßen monolithische UI's und Micro Frontends schwerpunktmäßig. Deshalb wurde im Rahmen der Aktionsforschung entschieden, eine bestehende monolithische UI in einem Microservice System zu einer Micro Frontend Lösung zu migrieren. Die Ergebnisse der Aktionsforschung unterstützen bei der Evaluierung der aus der Literaturanalyse und Case Study gewonnenen Erkenntnisse. Der nachfolgende Abschnitt stellt das Projekt JValue kurz vor und verdeutlicht den Aufbau durch die Microservice Architektur. Anschließend wird kurz auf die Funktionsweise der Microservices eingegangen.

JValue Project

JValue[1] ist ein Projekt um quelloffene Daten einfach nutzbar zu machen. JValue bietet dafür Schnittstellen an, um Extract-Transform-Load (ETL) Teilschritte auszuführen. Es ist eine microservice-basierte Anwendung, die im Rahmen der Aktionsforschung verwendet wird. Die folgende Sektion gibt einen kurzen Einblick in das System und deren Microservices.

Die JValue Anwendung[2] besteht aus fünf Kernkomponenten, die in eigene Microservices ausgelagert sind. Diese können unabhängig voneinander entwickelt und betrieben werden. Ein umfassendes Sequenzdiagramm, welches die Nachrichten und Beziehungen der Microservices darstellt, ist in Anhang A zu finden. Die Kommunikation der Dienste erfolgt über das Representational State Transfer (REST) Paradigma. Die nachfolgende Abbildung stellt die Microservice Architektur vereinfacht dar.

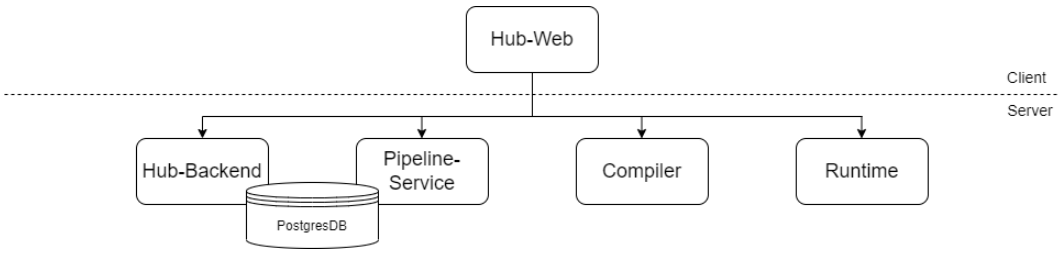

Abbildung 2.9: JValue Microservice Architektur

Der User kann über ein Web-Interface auf das System zugreifen. Das Frontend Hub-Web leitet die Anfragen des Users an die entsprechenden Microservices weiter. Über das Frontend kann der User Pipelines anlegen. Dieser kann bei der Konfiguration einer Pipeline unter anderem folgende Einstellungen festlegen:

- Datenquelle
- Anzahl Läufe (einmalig, stündlich, benutzerdefiniert)
- Auswahl der Runtime

[1] https://github.com/jvalue
[2] https://github.com/jvalue/hub

- Ziel des Datenimports

Wenn die Pipeline vom Hub-Backend und Pipeline-Service angelegt ist, kann der User einen Run initiieren. Dazu wird eine Verkettung von Service-Aufrufen vom Hub-Backend angestoßen. Dieser ruft den Pipeline-Service auf, welcher die Anfrage an den Compiler und an die Runtime weiterleitet. Diese führt den Datenimport von der in der Pipeline konfigurierten Quelle durch. Die importierten Daten werden anschließend an den Pipeline-Service und an das Hub-Backend weitergeleitet. Pipeline-Konfigurationen und zurückgelieferte Daten werden außerdem in einem zwischen Hub-Backend und Pipeline-Service verteilten PostgreSQL[3] Datenbanksystem persistiert.

Die Verwaltung und Entwicklung der Microservices baut auf dem Monorepo-Ansatz über GitHub auf. Dieser verwendet ein einziges Repository, um den gesamten Code für die Anwendung zu verwalten. Die Continuous Integration (CI) Pipeline der Anwendung läuft über GitHub Actions[4]. Diese baut die auf Node.js[5] basierten Microservices und erzeugt im Anschluss seperate Docker[6] Container. Zuletzt werden diese in die GitHub Container Registry[7] geladen und veröffentlicht.

JValue Hub-Web Frontend

Die Sektion stellt das Frontend Hub-Web kurz vor. Dieses ist für den End-Nutzer über den Webbrowser aufrufbar. Das Frontend ist eine monolithische UI in dem Microservice System. Der Service Hub-Web verantwortet das gesamte User Interface für JValue.

Der User Flow beschreibt die Gesamtheit aller Schritte die ein Nutzer beim Verwenden einer Website tun muss, um eine bestimmte Aktion auszuführen (Takahashi, 2016). Im Folgenden wird der User Flow des Frontends Hub-Web vereinfacht beschrieben. Der User beginnt dabei mit dem Login und beendet den Prozess mit dem manuellen Ausführen einer Pipeline. Im Anhang A.1 ist ein vereinfachtes Mock-Up des User Interfaces zu finden. Zusätzlich sind dort die Funktionalitäten der einzelnen Seiten dargestellt.

Beim Aufruf des User Interfaces gelangt der User zuerst auf die Landingpage. Dort kann er sich über eine Login Schaltfläche anmelden oder einen neuen Account erstellen. Auf der Landingpage sind zusätzlich eigene und fremde Projekte einsehbar. Ein Projekt beinhaltet im Kontext der JValue Anwendung zugehörige Pipelines und deren Versionen. Außerdem kann ein angemeldeter User auf der Landingpage Projekte anlegen und verwalten.

[3] https://www.postgresql.org/
[4] https://github.com/features/actions
[5] https://nodejs.org/en/
[6] https://www.docker.com/
[7] https://github.com/features/packages

Dazu wird er auf die ProjectPage weitergeleitet. Auf dieser kann ein User Pipelines und deren Versionen verwalten. Jede Pipeline, Version und Projekt sind einer eindeutigen ID zugeordnet.

Zum Bearbeiten und Starten einer Pipeline wird der User auf die PipelinePage geleitet. Dort kann er zudem die über die Pipeline extrahierten Daten einsehen. Damit endet der User Flow.

Eignung der Microservice Anwendung für Aktionsforschung

Der Absatz stellt kurz die Gründe für die Auswahl der Anwendung für die Aktionsforschung auf.

In erster Linie relevant ist, dass die Anwendung aus Microservices aufgebaut ist. Die Anzahl der Microservices ist für den Kontext dieser Aktionsforschung nicht relevant. Es bedarf lediglich der Anforderung, dass die Anwendung ein monolithischen User Interface integriert. Dieses wird durch das Frontend Hub-Web abgebildet und deren Migration soll im Rahmen der Aktionsforschung untersucht werden.

Um Guidelines abzuleiten, soll die Migration prototypisch anhand der Anwendung vorgenommen werden. Für die Umsetzung ist deshalb von Bedeutung, dass die Anwendung quelloffen ist und veränderbar sein darf. JValue ist ein Projekt, welches vom Lehrstuhl für Open-Source Software an der Friedrich-Alexander Universität Erlangen Nürnberg begleitet wird. Damit wird das Kriterium erfüllt.

2.5.2 Micro Frontend Design

Dazu wird das existierende Frontend Hub-Web von JValue verwendet. Nachfolgend wird das Konzept und die Umsetzung beschrieben.

Das Ziel der Aktionsforschung ist die Herauslösung einer Komponente als Micro Frontend. Die Komponente, in diesem Fall die PipelinePage soll dabei unabhängig als Anwendung ausführbar und entwickelbar sein. Abbildung 2.10 zeigt das Konzept und das Zusammenspiel der Anwendungen in der Micro Frontend Umgebung.

Das Frontend des Hub-Webs agiert als Container. Es beinhaltet alle weiteren Komponenten zur Darstellung der LandingPage und der ProjectPage. Das Frontend ist mithilfe von React[8] realisiert. Um Zustände gegenseitig auszutauschen, greifen die Komponenten des Frontends Hub-Web auf einen gemeinsamen Redux Store[9] zu. Darin werden zum Beispiel Informationen zum User und Queries über Pipelines und deren Runs abgelegt. Die Komponenten können bei Bedarf auf diese zugreifen.

[8] https://reactjs.org/
[9] https://redux.js.org/

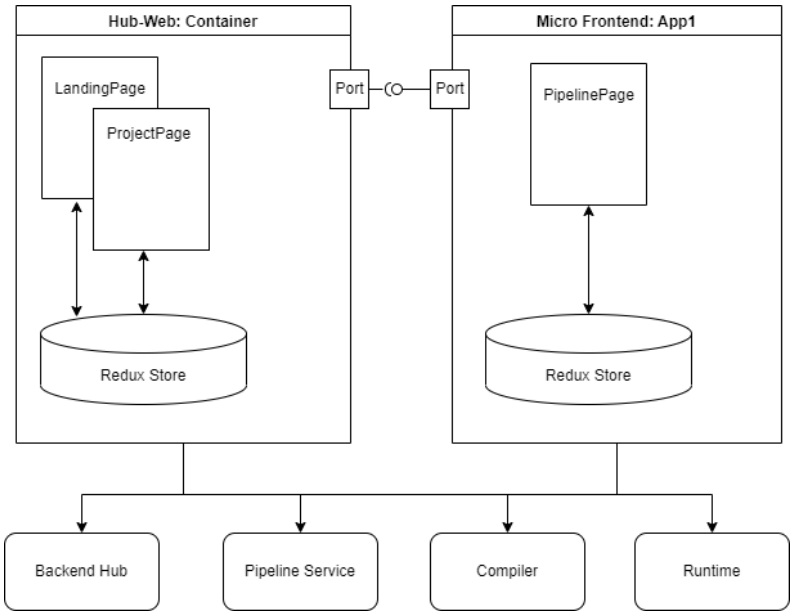

Abbildung 2.10: Micro Frontend Konzept:
Herauslösen der PipelinePage Komponente

Die PipelinePage wird als eingeständige Komponente aus dem Frontend Hub-Web herausgelöst und als eigene Anwendung gestartet. Sie wird über einen Port freigegeben. Das Frontend Hub-Web agiert dabei als Container und lädt die Inhalte aus dem extrahierten Micro Frontend über den Port zur Laufzeit nach. Des Weiteren greift die PipelinePage auf einen seperaten Redux Store zu.

Zuletzt kommunizieren beide Teil-Anwendungen mit den Backend-Services und deren API. Für ein funktionierendes Produktivsystem müssen diese gestartet sein, zur Entwicklung reicht ein unabhängiges Starten der jeweiligen Anwendung aus.

Iterationen

Um bestehende Konzepte zu verfeinern und Erkenntnisse zu bündeln wurde iterativ vorgegangen. Nachfolgend sind die durchgeführten Iterationen abgebildet.

In der ersten Iteration wurde die gesamte Komponente herausgelöst und integriert. Dieses Big Bang Vorgehen zeichnete sich als unpraktisch aus, weshalb in der zweiten Iteration nur ein Minimalbeispiel der PipelinePage-Komponente migriert wurde. Zentrale Abhängigkeiten wie Redux Stores oder REST-Aufrufe ans Backend wurden entfernt, um Fehler bei der Integration zu minimieren. Nachdem das Micro Frontend System lauffähig war, wurde die Komponente in den nächsten Iterationen um die fehlenden Funktionalitäten erweitert.

Die dritte Iteration setzt eine funktionierende Micro Frontend Landschaft voraus. In diesem Schritt hat sowohl das Micro Frontend als auch die Container Anwen-

No.	Beschreibung	Technologien
1	Herauslösen der gesamten Komponente	React, Webpack
2	Herauslösen der Komponente mit minimaler Funktionalität	React, Webpack
3	Integration mithilfe eines seperaten Redux-Stores	React, Redux
4	Auslagerung von zentralen Abhängigkeiten	React, Redux
5	Integration von Lazy Loading und Fehlerbehandlung	React

Tabelle 2.12: Iterationen der Aktionsforschung

dung einen seperaten Redux Store. Dieses Konzept ist ebenfalls in Abbildung 2.10 dargestellt. Da dies Framework-spezifische Abhängigkeiten mit hoher Kopplung sind, werden diese Teile in der vierten Iteration als Properties ausgelagert und über das Interface zur Verfügung gestellt. Das erhöht die Wiederverwendbarkeit der Komponente.

In der letzten Iteration wurden Lazy Loading Mechanismen umgesetzt und Fehlerbehandlung bei einem Ausfall des Micro Frontends eingebaut.

2.5.3 Implementierung

Im ersten Schritt wird die Anwendung geändert, sodass diese mit Webpack[10] gestartet werden kann. Anschließend wird ein zweites Repository erzeugt und darin mithilfe von create-react-app[11] ein React Framework aufgesetzt. Die Komponente der PipelinePage wird in das zweite Repository migriert und seperat über Webpack ausführbar gemacht.

Webpack ermöglicht das Bündeln von JavaScript Quellcode zu einer einzigen ausführbaren JavaScript Datei. Über Webpack können die erzeugten Bundles außerdem über das Module Federation Plugin[12] über einen Port freigegeben werden. In der Container-Anwendung Hub-Web wird daraufhin ebenfalls über Module Federation das freigegebene Bundle über ein Remote eingebunden. Das ermöglicht die Integration der freigegebenen Komponente in die Container-Anwendung zur Laufzeit. Damit Webpack die Dateien asynchron nachladen kann, müssen die Anwendungen außerdem Bootstrapping integrieren. In React bezeichnet dies einen Mechanismus, welcher das gesamte Projekt in einer initialen Komponente über eine Funktion importiert.

[10] https://webpack.js.org/
[11] https://create-react-app.dev/
[12] https://webpack.js.org/concepts/module-federation/

2.5.4 Erkenntnisse

Das Kapitel zeigt die gewonnenen Erkenntnisse aus der durchgeführten Aktionsforschung. Anschließend wird daraus eine technische Guideline abgeleitet, welche die Migration einer Komponente hin zu einem Micro Frontend vereinfachen soll. Die nachfolgenden Tabellen stellen eine Auflistung dar. Dabei wird unterteilt in Bestärkung der Theorie und neue Erkenntnisse. Dadurch wird ersichtlich, welche Punkte bereits in der strukturierten Literaturanalyse oder in der Case Study behandelt und dadurch bestätigt wurden.

No.	Kategorie	Beschreibung
A-V1	Vorteil	Fehlertoleranz
A-V2	Vorteil	Unabhängige Entwicklung und Deployment
A-N2	Nachteil	Umständliches Debugging
A-N3	Nachteil	Schwierigere Fehlerbehandlung
A-N4	Nachteil	Erhöhte Komplexität

Tabelle 2.13: Aktionsforschung: Bestärkung der Theorie

Durch das Herauslösen der Komponente als eigenständige Anwendung, erhöht sich die Fehlertoleranz **(A-V1)**. Die Anwendungen können auf unterschiedlichen Servern ausgeführt werden, was für ein robusteres Gesamtsystem sorgt **(A-V2)**. Das Debugging wird durch die Ausführung auf verschiedenen Systemen ebenfalls erschwert. Zusätzlich kommen bei der Integration weitere Fehlerquellen hinzu, was die Entwicklung weiterhin verlängert **(A-N2)**. So kann die Container-Anwendung Hub-Web auch lauffähig sein, wenn die PipelinePage gerade im Wartungsmodus und damit nicht verfügbar ist. Die Lazy Loading Technologie von React unterstützt dabei bei der Fehlerbehandlung, ist aber komplexer zu integrieren **(A-N3)**. Auch die generelle Komplexität der Anwendung ist durch die Integration der Micro Frontends gestiegen **(A-N4)**.

No.	Kategorie	Beschreibung
A-N1	Nachteil	Gemeinsamer Zustand
A-H1	Herausforderung	Zentrale Abhängigkeiten

Tabelle 2.14: Aktionsforschung: Neue Erkenntnisse

Die Anwendungen in einer Micro Frontend Umgebung agieren autonom. Deshalb ist es ohne Weiteres nicht möglich, einen gemeinsamen Zustand herzustellen **(A-N1)**. React bietet mit Redux einen zentralen Platz für Zustände, dieser ist jedoch auf ein einziges Micro Frontend beschränkt. Das Erzeugen eines gemeinsamen Zustands und das Verwalten zentraler Abhängigkeiten stellt sich somit

als Herausforderung dar **(A-H1)**. Wenn Zustände synchronisiert werden sollen, muss dies über eine gemeinsame API oder ein Messaging-System erfolgen. Das stellt einen Nachteil von Micro Frontends dar und erhöht zudem die Komplexität der Anwendung.

Insgesamt bestätigt sich das Gesamtbild aus der strukturierten Literaturanalyse und der Case Study hinsichtlich Micro Frontends.

2.5.5 Leitfäden für die Migration

Die Migration einer Komponente ist ebenfalls mit Nachteilen und Herausforderungen verbunden. Nachfolgende technische Leitfäden unterstützen bei der Wiederverwendbarkeit der Komponente und können Praktikern bei der Migration helfen. Die Leitfäden haben sich aus den Erkenntnissen der Aktionsforschung gebildet und sind nicht vollständig erprobt.

(1) Auslagerung zentraler Abhängigkeiten als Komponenten-Attribute

Wenn es das Micro Frontend erfordert, dass Zustände zentral bereitgestellt werden, wird die Wiederverwendbarkeit der Komponente verringert. Aus der Aktionsforschung hat sich die Erkenntnis gebildet, dass diese Abhängigkeiten bei der Migration in Properties ausgelagert werden sollten. Im Beispiel verwendet das Micro Frontend einen eigenen Redux Store. Wenn das Micro Frontend anschließend in die Container Anwendung integriert wird, muss dieses also ebenfalls den identischen Redux Store einsetzen. Damit ist es nicht mehr möglich, dass der Container auf einem anderen Framework basiert, als das Micro Frontend. Dies wirkt sich somit negativ auf die Wiederverwendbarkeit der Micro Frontend Anwendung aus.

Aus diesem Grund werden in Guideline (1) zentrale Abhängigkeiten, unter anderem Redux Store Zugriffe oder Queries, die zum Framework gehören im Micro Frontend als Properties ausgelagert.

(2) Bereitstellung des Micro Frontends mit Komponenten-Attributen als Interface

Diese werden anschließend in Guideline (2) als Interface zur Verfügung gestellt. Container Anwendungen, die das Micro Frontend verwenden, müssen beim Aufruf des Frontends also die Properties übergeben.

(3) Einbindung einer Wrapper Komponente in aufrufende Anwendungen

Um Komponenten-Attribute beim Aufruf des Micro Frontends zu übergeben, eignet sich eine Wrapper Komponente, um diese Funktionalitäten zu kapseln. Im Beispiel der Aktionsforschung führt die Wrapper Komponente die Zugriffe

auf Redux aus und ruft das Micro Frontend mit den benötigten Properties auf. Abbildung 2.11 stellt das finale Konzept von Guideline (3) unter Verwendung der Wrapper Komponente dar.

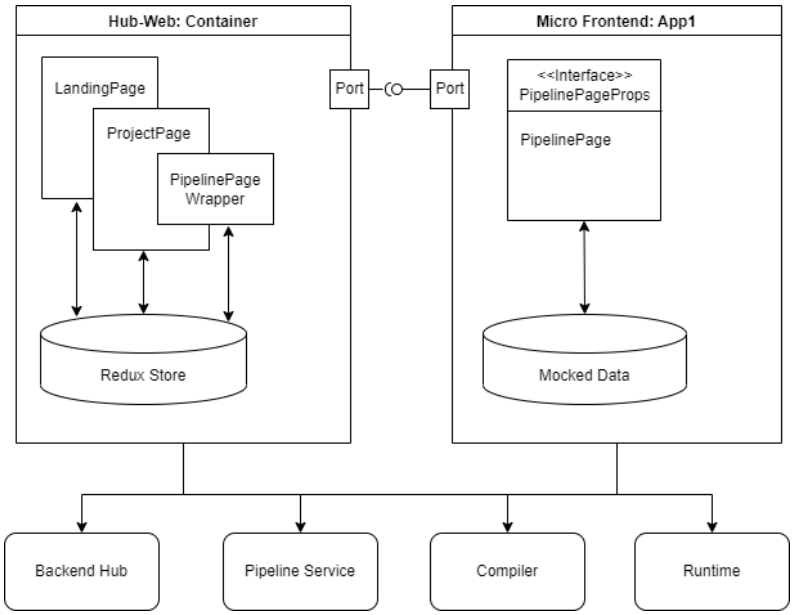

Abbildung 2.11: Finales Micro Frontend Konzept:
Interface-Erzeugung und Wrapper Komponente

(4) Integration von Lazy Loading

Webpack stellt Funktionalitäten bereit, um mehrere JavaScript Dateien zu einer Datei zu bündeln. Die erzeugten Bundles können bei größeren Micro Frontends, die viele Bibliotheken einbinden, auf mehrere Megabytes anwachsen. Werden Micro Frontends über konventionelle Import-Mechanismen integriert, so lädt die Container-Anwendung die Micro Frontend Bundles mit. Mithilfe von Lazy Loading kann der Inhalt zu der Zeit geladen werden, wenn er benötigt wird (Wickham, 2018). Somit verbessert Guideline (4) die Performance und das Benutzererlebnis der Anwendung. Außerdem können mithilfe von Webpack Bibiliotheken, die sowohl von Micro Frontend als auch von Container Anwendungen verwendet werden, gemeinsam nutzbar gemacht werden.

2.6 Diskussion und Ausblick

Das Kapitel fasst die Ergebnisse aus den vorherigen Teilbereichen kurz zusammen und stellt Schlussfolgerungen auf. Anschließend werden Limitationen der Arbeit aufgezeigt und zur Diskussion gestellt. Zum Schluss wird ein Ausblick auf die weitere Forschung gegeben.

Sowohl die Literaturanalyse als auch die Case Study zeigen, dass es nur wenige Integrationslösungen für User Interfaces in microservice-basierten Systemen gibt. Es überwiegen Forschungsartikel über monolithische User Interfaces und Micro Frontends. Dahingegen existieren nur wenige Daten über User Interfaces in SCS. Je nach Anforderung der Anwendung können SCS jedoch ein Zielzustand der Anwendungsarchitektur sein.

Zudem stellt sich heraus, dass Micro Frontends nicht in jedem System sinnvoll genutzt werden können. Größere Datenübertragungen können das Benutzererlebnis beeinträchtigen und die höhere Komplexität führt dazu, dass Micro Frontends gerade für kleinere Teams keinen wesentlichen Mehrwert bringen.

Der Einsatz von Micro Frontends kann sinnvoll sein, wenn die Microservices des Backends anhand der User Journey geschnitten sind. Jedoch existieren nur wenige Guidelines für die Migration von bestehenden Komponenten hin zu Micro Frontends. Die Ergebnisse der Aktionsforschung sollen bei der Migration unterstützen.

Limitationen

Jedoch muss berücksichtigt werden, dass die durchgeführte Aktionsforschung nur einen exemplarischen kleinen Teil abdeckt. Das in der Arbeit verwendete Web-Framework ist React. Die Migration von Komponenten weiterer Web-Frameworks aus unterschiedlichen Anwendungen kann aufbauend auf dieser Arbeit untersucht werden.

Die strukturierte Literaturanalyse wurde mit einem systematisch entwickelten Suchbegriff durchgeführt. Dieser ist aufbauend auf Kombinationen bestimmter Begriffe aus dem Kontext der Arbeit. Diese Eingrenzung kann weitere Werke ausschließen, die aber relevante Erkenntnisse bringen können. Eine Anpassung des Suchbegriffs ermöglicht weitere Forschung.

Ausblick

Ein kurzer Abgleich zwischen Literaturanalyse und Case Study zeigt auf, dass nur wenige Punkte identisch sind. Auch ergeben sich zum Beispiel bei den Punkten Wartbarkeit oder Performance Widersprüche. Daraus lässt sich schlussfolgern, dass weitere Forschung in dem Themengebiet notwendig ist.

Um die Validität der aus der Case Study generierten Daten zu erhöhen, können weitere Experteninterviews, basierend auf dem verwendeten Interview Guide durchgeführt werden. Die durchgeführte Case Study bezieht sich lediglich auf ein Unternehmen. Die Sektion Convenience der Case Study erläutert Gründe für die Auswahl des Unternehmens. Aufbauend darauf können weitere Fälle untersucht werden.

Die Ergebnisse der Aktionsforschung beinhalten Guidelines für die Migration hin

zu einer Micro Frontend Architektur. Aus der strukturierten Literaturanalyse geht hervor, dass es kaum Literatur gibt, die Migrationen von User Interfaces in Microservices hin zu Micro Frontends behandelt. Diese Fälle können in der weiteren Forschung untersucht werden und weitere Guidelines entwickelt werden. Somit können die in der Arbeit vorgestellten Ergebnisse als Ansatz für die weitere Forschung aufgefasst werden.

3 Weiterführende Erläuterungen

Zu Beginn wird ein kurzer Überblick über die in der strukturierten Literaturanalyse verwendete Literaturbasis gegeben. Dort sind unter anderem Begründungen für die Auswahl zu finden. Anschließend erläutert das Kapitel technische Inhalte und Konzepte der Arbeit. Es werden die im Zuge der Aktionsforschung verwendeten Technologien und Konzepte dargelegt und veranschaulicht.

3.1 Ausgangspunkt

3.1.1 Microservices

Microservice-basierte Systeme setzen sich aus einzelnen eigenständigen Services zusammen, die über Netzwerkmechanismen miteinander kommunizieren. Diese sind unabhängig voneinander lauffähig und entwickelbar. (Newman, 2021)

Im Unterschied zu monolithischen Architekturen, sind microservice-basierte Systeme lose gekoppelt. Dadurch beeinflussen Änderungen an einzelnen Diensten andere Services nicht. Dies führt zu einer besseren Skalierbarkeit des Gesamtsystems und trägt dazu bei, dass sich die Organisationsstruktur der Unternehmen gut an die Aufteilung der Services anpassen kann. Außerdem kann somit die Wartbarkeit des Gesamtsystems verbessert werden. (Wolff, 2018)

Um mit anderen Services zu kommunizieren, stellen die Microservices in der Regel eine Application Programming Interface (API) zur Verfügung. Dies sind in der Praxis oft RESTful-Schnittstellen, über die die Microservices ihre Funktionalitäten veräußern können. REST stellt dabei ein Programmierparadigma dar, um verteilte Systeme zu entwickeln. Es basiert auf dem HTTP-Protokoll und verwendet das Prinzip der Zustandslosigkeit. HTTP-Nachrichten zwischen Sendern und Empfänger laufen isoliert voneinander ab, es werden somit von keiner Seite Inhalte zwischengespeichert. (Richardson & Ruby, 2008)

Außerdem werden zur Kommunikation häufig sogenannte Messaging-Systeme verwendet, die unabhängig von den Microservices Funktionalitäten zum Austausch von Nachrichten anbieten. Anders als bei RESTful-APIs implementieren diese das

Advanced Message Queuing Protocol (AMQP) Protokoll. Die meisten Messaging Systeme basieren auf dem Publisher-Subscriber Prinzip. Ein Publisher sendet Nachrichten mit einer bestimmten Topic an einen Message Broker. Dieser leitet die Nachricht an einen oder mehrere Subscriber weiter, die sich auf das Topic registriert haben. In der Praxis haben sich daraus Systeme wie RabbitMQ oder Apache Kafka etabliert. (Magnoni, 2015)

Im Kontext von Microservices bieten beide Varianten zur Kommunikation unterschiedliche Vorteile. Die Kommunikation über RESTful-Schnittstellen erfolgt synchron, während ein Nachrichtenaustausch über AMQP asynchron verläuft. In der Praxis werden oft verschiedene Verfahren kombiniert. (Fernandes et al., 2013)

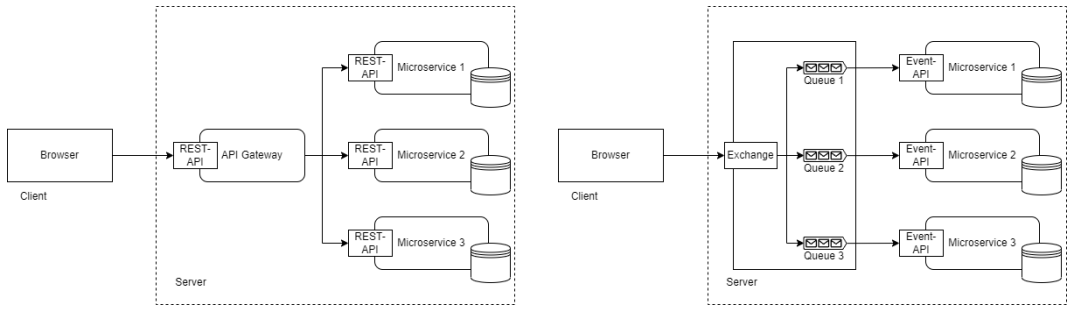

Abbildung 3.1: Vergleich Kommunikation von Microservices:
RESTful-Webservices und Messaging-Systeme
In Anlehnung an Hong et al. (2018)

3.1.2 Bounded Context

Unternehmen stellt die Aufteilung der Anwendung in Microservices häufig vor eine Herausforderung. Soll eine Microservice Architektur verwendet werden, ergeben sich grundsätzliche Fragestellungen:

- Welcher Service verantwortet welche Funktionalität?
- Welches Team verantwortet welche Services?

Ein Ansatz, um diese Herausforderungen zu bewältigen ist Domain-Driven Design (DDD). Mithilfe von DDD können Anwendungen anhand von Domänen modelliert und in Microservices geschnitten werden. Beim Entwurf der Anwendungen werden Domänen identifiziert und in sogenannte Bounded Contexts getrennt. Diese verstehen sich als Cluster für kohärente Teile einer Domäne. Die Modellierung kann dabei zum Beispiel über Unified Modeling Language (UML) vorgenommen werden. Ein Bounded Context begrenzt den Einsatzbereich eines Domänenmodells und umfasst die Business-Logik für eine bestimmte Fachlichkeit. (Rademacher et al., 2018)

Die Identifizierung der Bounded Contexts unterstützt bei der Aufteilung der Anwendung in Microservices. Die Anwendung wird entlang der Bounded Contexts in Microservices geschnitten. Darüber hinaus können Beziehungen zwischen den Domänen als Kommunikationsschnittstellen modelliert werden. (Rademacher et al., 2018)

Die Ergebnisse der Case Study zeigen, dass der Backend-Schnitt der Anwendung Einfluss auf die Einsatzmöglichkeiten von Micro Frontends hat. Sind die Microservices nicht entlang der User Journey geschnitten, ist die Integration von Micro Frontends mit weiteren Herausforderungen verbunden.

3.1.3 Organisationsstrukturen

Conveys Law hat sich dazu als Best Practice bewährt. Diese besagt, dass sich die Organisation der Entwicklungsteams nach der Architektur der Anwendung richten soll und nicht umgekehrt. (Conway, 1968)

Im Kontext von Microservices bedeutet das, dass die Verantwortungsbereiche der Entwicklungsteams ebenfalls nach Microservices getrennt und definiert werden. In der Regel verantwortet ein Team einen Microservice. Damit wird den Teams eine Ende zu Ende Verantwortlichkeit ermöglicht und sie agieren somit crossfunktional. Es sollte nicht vorkommen, dass mehrere Entwicklungsteams einen Microservice übernehmen. (Atchison, 2016)

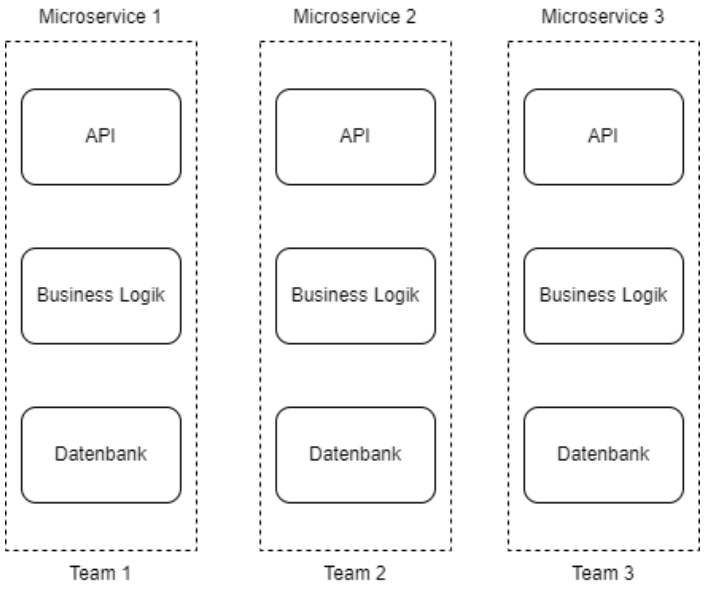

Abbildung 3.2: Ausrichtung der Entwicklungsteams nach Microservices Gehani (2022)

3.2 Technologien

3.2.1 React

React ist eine JavaScript Open-Source Bibliothek, um User Interfaces zu entwickeln. Das Web-Framework etabliert Konzepte wie virtuellen DOM und unidirektionale Datenflüsse und unterstützt bei der Entwicklung von performanten Single Page Application (SPA). Die Sektion erläutert die Konzepte des Frameworks kurz.

Wenn eine Webseite im Browser geladen wird, erstellt dieser ein DOM der Webseite. Das DOM stellt die Webseite in einer Baumstruktur aus HTML-Tags dar. Eine SPA ist im Gegensatz zu traditionellen Web-Anwendungen nicht mehr Linkbasiert, sondern der gesamte Inhalt ist in einer Webseite eingebunden. Damit der Browser auf Events reagieren kann und Änderungen der Webseite anzeigen kann, muss somit das DOM zur Laufzeit manipuliert werden. (Fedosejev, 2015)

DOM Manipulationen sind in der Regel sehr langsam. Um dem entgegen zu wirken nutzt React einen virtuellen DOM. Dieser ist eine Representation des DOM, die im Arbeitsspeicher gehalten wird. Anstelle die gesamte Seite bei einer Änderung neu zu laden, berechnet React den Unterschied zwischen dem vorhergehenden DOM und dem neuen DOM. Somit wird nur die tatsächliche Differenz im Browser neu geladen. (Fedosejev, 2015)

Um Daten zwischen View und Model zu synchronisieren existieren zwei prinzipielle Möglichkeiten. Während Web-Frameworks wie Angular bidirektionale Datenflüsse verwenden, gibt es bei React einen unidirektionalen Datenfluss zwischen Komponenten. Daten können lediglich von übergeordneten Komponenten zu deren Unterkomponenten transferiert werden. Die Argumente, die eine Komponente beim Aufruf durch die übergeordnete Komponente erhält, werden in React als Props bezeichnet. Somit ist es zuerst nicht möglich, Daten von untergeordneten Komponenten nach oben hin weiter zu geben. Dies ist jedoch ein in der Praxis oft auftretendes Szenario. Damit das trotzdem funktioniert, existieren in React Callback-Funktionen, die über Events aufgerufen werden. (Gackenheimer & Paul, 2015)

3.2.2 Webpack

Webpack ist ein Open-Source Werkzeug in der Web-Entwicklung. Es ermöglicht das Bündeln von JavaScript Modulen zu sogenannten statischen Bundles. Diese können von Webservern an Clients ausgeliefert werden. Dabei erlaubt Webpack das Bündeln von verschiedenen, in der Web-Entwicklung gängigen Formaten. Außerdem werden die Abhängigkeiten der Module aufgelöst und in der Ergebnisdatei integriert. Diese besteht nun aus einer einzigen JavaScript Datei. (Zammetti,

2020)

Die Funktionsweise ist in der folgenden Abbildung vereinfacht dargestellt.

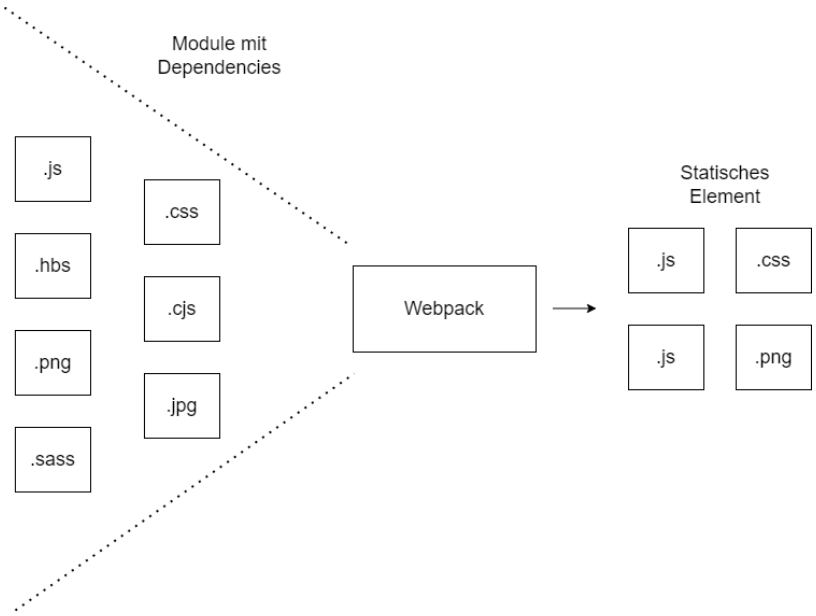

Abbildung 3.3: Webpack Funktionsweise („Webpack“, 2022)

Webpack kann über die Konfiguration in der Datei webpack.config.js verwaltet werden. Darin werden Einstellungen wie Entry-Einträge und Output-Dateien vorgenommen. Um nötige Dependencies aufzulösen, erzeugt Webpack Dependency-Diagramme, deren Wurzel in den Entry-Einträgen beschrieben sind. (Zammetti, 2020)

Zusätzlich können in Webpack Loader konfiguriert und angewendet werden. Loader sind notwendig, damit Webpack die verschiedenen Dateiformate transpilieren und verarbeiten kann. (Zammetti, 2020)

Abschließend können darin Plugins spezifiziert werden, welche weitere Funktionalitäten bieten. Module Federation ist ein Beispiel für ein Plugin, welches im nachfolgenden Kapitel beschrieben wird. (Zammetti, 2020)

3.2.3 Module Federation

Das Module Federation Plugin in Webpack bietet die Möglichkeit, die erzeugten Bundles auf dem ausführenden System über ein Port freizugeben. Dies hat den Nutzen, dass andere, mit Webpack ausgeführte Anwendungen, das Bundle integrieren können. Das Einbinden der freigegebenen Datei funktioniert über den Remote-Eintrag in der Konfigurationsdatei von Webpack. Anschleßend kann die

Komponente zum Beispiel über den Lazy-Import Mechanismus von React geladen werden. Die folgenden Konfigurationen zeigen das Freigeben und Integrieren der PipelinePage Komponente mithilfe von Webpack.

Modul app1 gibt gebündelte Komponenten der PipelinePage frei:

```
module.exports = {
  plugins: [
    new ModuleFederationPlugin({
      name: 'app1',
      filename: 'remoteEntry.js',
      exposes: {
        './PipelinePage':
            './src/PipelinePage/PipelinePage',
      },
    }),
  ],
};
```

Modul PipelinePage wird in weiterer Anwendung integriert:

```
module.exports = {
  plugins: [
    new ModuleFederationPlugin({
      name: 'container',
      filename: 'remoteEntry.js',
      remotes: {
        app1:
            'app1@http://localhost:3001/remoteEntry.js',
      },
    }),
  ],
};
```

Module Federation bietet somit die Grundlage, um eine Micro Frontend Umgebung zu entwickeln. Es erlaubt zusätzlich den parallelen Einsatz verschiedener Frameworks. Außerdem werden über Lazy Loading Ansätze beim Integrieren des Remotes Performanceverbesserungen erreicht. (Taibi & Mezzalira, 2022)

Der Import des Remotes ist abhängig vom verwendeten Web-Framework. In React wird das PipelinePage Remote über nachfolgenden Mechanismus zur Laufzeit geladen:

```
const PipelinePage = React.lazy(
    () => import('app1/PipelinePage')
);
```

Dies beinhaltet einen Lazy Import, sodass die aufrufende Komponente geladen werden kann, ohne dass das Micro Frontend verfügbar ist. Sobald das Micro Frontend vollständig über das Netzwerk empfangen wird, lädt es die React Anwendung in die aufrufende Komponente.

Außerdem ist es über den Lazy-Import Mechanismus von React möglich, ein Error Boundary zu definieren. Dies wird als Fallback verwendet, falls das Micro Frontend nicht verfügbar ist.

```
<ErrorBoundary>
    <Suspense fallback={<div>Loading...</div>}>
        <PipelinePage
          params={params}
          user={user}
          showOwnRuns={showOwnRuns}
          pipeline={pipeline}
          project={project}
          allRunIds={allruns}
          initRun={initRun}>
        </PipelinePage>
    </Suspense>
</ErrorBoundary>
```

3.2.4 Redux

Redux ist eine Open-Source JavaScript Bibliothek, welche Zustände verwalten und synchronisieren kann. Redux besteht aus einem einzigen Store, welcher alle Zustände der Anwendung enthält. Komponenten können den Zuständ bei Bedarf lesen. Es basiert auf drei grundlegenden Prinzipien, die im Folgenden kurz erläutert werden.

Das erste Prinzip leitet sich aus dem Prinzip "Der einzige Punkt der Wahrheitäb. Es besagt, dass es einen Datenbestand gibt, der korrekt und zuverlässig ist. Dieses Prinzip wird in Redux über einen einzigen Store abgebildet. Dies vereinfacht die Entwicklung, da der Zustand an einem einzigen zentralen Ort abgelegt wird. (Bugl, 2017)

Des Weiteren baut Redux darauf aus, dass der Zustand nur lesbar und nicht direkt änderbar sein soll. Um Zustände zu ändern, wird stattdessen bei der Initialisierung eine Aktion übergeben, unter welchen Voraussetzungen sich der Zustand ändern soll. (Bugl, 2017)

Zuletzt lässt Redux Zustandsänderungen lediglich über pure Funktionen zu. Als pure Funktionen werden Funktionen bezeichnet, deren Rückgabewert ausschließlich von den Eingabeparametern abhängt und die keine Nebenwirkungen auf-

weist. In Redux werden diese als Reducer angegeben. Sie erhalten den vorherigen Zustand und eine Aktion als Eingabeparameter und geben den neuen Zustand zurück. (Bugl, 2017)

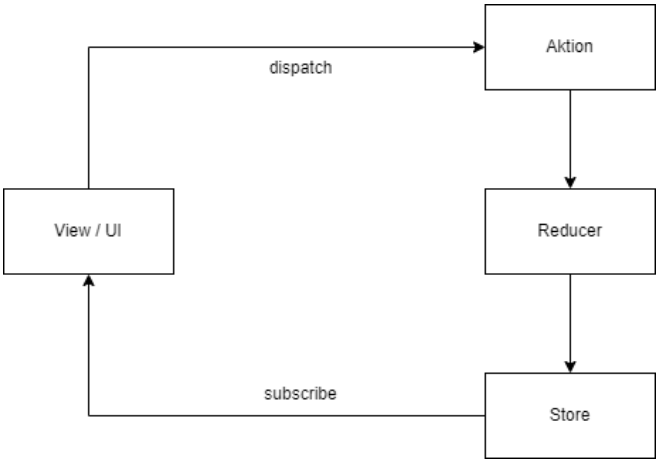

Abbildung 3.4: Redux - Grundlegende Prinzipien

Im Frontend Hub-Web wird Redux zum Beispiel dafür verwendet, den Zustand des angemeldeten Users abzulegen und für andere Komponenten verfügbar zu machen. Als Aktion zur Änderung des Zustands wird dabei zum Beispiel der Zugriff auf die Backend-API für User Authentifizierung verwendet. Sobald eine Komponente einen Aufruf tätigt, wird implizit der Zustand innerhalb der Anwendung entsprechend angepasst.

Die Ergebnisse der Aktionsforschung stellen dar, dass Zustände die über Redux verwaltet werden, als Props ausgelagert werden sollten. Um möglichst generalisiert und wiederverwendbar zu sein, sollten Zustände nicht im Micro Frontend verwaltet werden. Wenn es das Micro Frontend erfordert, dass ein Redux Store verwendet wird, dann muss die Container Anwendung ebenfalls den identischen Redux Store einsetzen. Dies ist deshalb so, da das Micro Frontend zur Laufzeit in die Container Anwendung geladen wird. Das führt zu einer engen Kopplung zwischen Micro Frontend und Container Anwendung und verringert die Wiederverwendbarkeit.

Appendices

A JValue Microservice Architektur

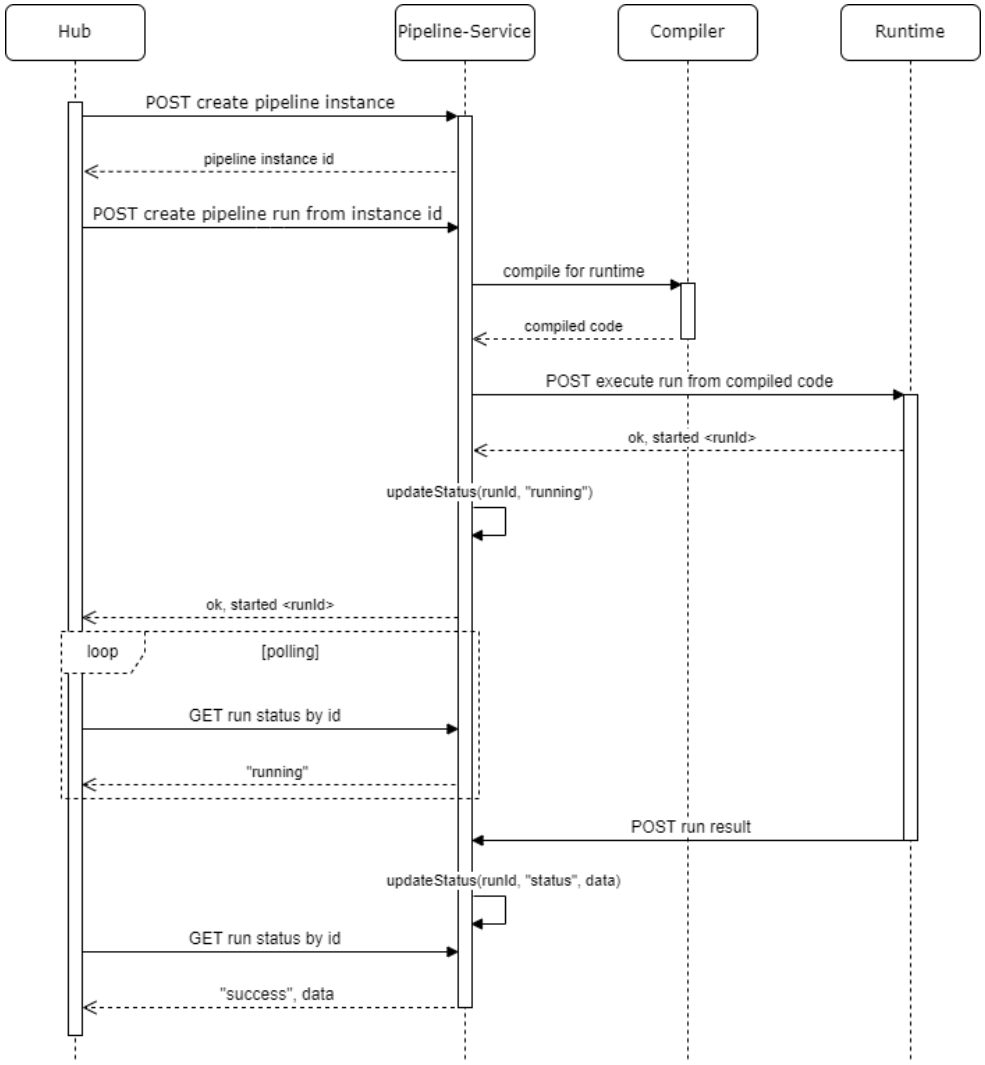

Abbildung A5: Sequenzdiagramm der JValue Microservices. Der Service Hub enthält sowohl die Funktionalitäten der Services Web-Hub und Backend-Hub. Als Runtime steht zum Zeitpunkt der Arbeit eine runtime-simple zur Verfügung. Weitere Runtimes wie Kafka-Server sind in der Entwicklung.

A.1 JValue Frontend

Abbildung A6: Vereinfachtes User Flow Diagramm des Frontends Hub-Web. Es zeigt den User Flow vom Aufruf der LandingPage bis zum Anlegen und Ausführen einer ETL-Pipeline.

B Strukturierte Literaturanalyse

No.	Autor	Titel	Auswahl
1	Harms Holger Rogowski Collin	Guidelines for adopting frontend architectures and patterns in microservices-based systems	Ja
2	Li Kunying Ding Yu Shen Duanming Li Qing Zhen Zebing	The Design and Research of Front-End Framework for Microservice Environment	Ja
3	Tilak P Yedhu Yadav Vaibhav Dharmendra Shah Dhruv Bolloju Narasimha	A platform for enhancing application developer productivity using microservices and micro-frontends	Ja
4	Yang Yuan	Research and Implementation of Web Front End Development System Based on Microservice	Ja
5	Fernández-Garcıa Antonio Jesus	A Recommender System for Smart User Interfaces using Machine Learning and Microservices	Nein
6	Bruno Leonel André Lopes	Data Science for Non-Programmers: Orchestration of Microservices and Graphical User Interface	Nein
7	Prajwal Y.R. Parekh Jainil Viren Shettar Rajashree	A Brief Review of Micro-frontends	Ja
8	Peltonen Seviri Mezzalira Luca Taibi Davide	Motivations, benefits, and issues for adopting Micro-Frontends	Ja

Das Kapitel stellt die für die strukturierte Literaturanalyse verwendete Literatur kurz vor. Die abschließende gesammelte tabellarische Aufbereitung der Datensammlung ist in Anhang B angefügt.

Datensammlung

Harms et al. (2017) zeigen in dem Artikel drei Strategien auf, um Frontend-Integration in Microservices durchzuführen. Dabei wird zwischem monolithischen User Interface, Micro Frontends und Self-Contained-Systems unterschieden. Für die genannten Strategien werden daraufhin Vorteile und Nachteile aufgezeigt. Im Rahmen des Artikels werden die Ansätze prototypisch bei einer äquivalenten Anwendung implementiert und evaluiert. Aus den Ergebnissen des Vergleichs werden bewährte Praktiken und Richtlinien bei der Frontend-Integration dargestellt.

In dem Conference Paper zeigen Li et al. (2020) ein Framework auf, mit dem sich User Interfaces in Microservices umsetzen können. Der Fokus des Frameworks liegt dabei auf der Komponentisierung des Frontends. Damit soll eine bessere Skalierbarkeit und Effizienz der Anwendung erreicht werden.

Die Publikation von Tilak et al. (2020) thematisiert Microservices und Micro-Frontends. Als wesentliche Kernfrage wird herausgearbeitet, wie es möglich sein kann, dass viele Teams einer Organisation verschiedene Microservices und Micro Frontends unabhängig voneinander entwickeln und warten können. Die Autoren stellen dafür eine Plattform vor, die Entwickler dabei bei Kommunikation und Transparenz unterstützen soll.

Yang (2022) behandelt Web-Plattform Frontends. Die Zielsetzung des Artikels ist es, die Entwicklung und das Design von Web-Plattformen im Kontext von Microservices zu verbessern. Dazu schlägt Yang (2022) ein Framework vor, welches ähnlich zu dem MVC Framework ist.

MVC bezeichnet ein Entwurfsmuster, welches die Aufteilung der Anwendung in drei Komponenten vorsieht. Das Model übernimmt das Halten des Datenmodells, während das View die Inhalte des Models als User Interface visualisiert. Der Controller stellt das Interface zwischen View und Model dar. Er gibt Änderungen am Model an das View weiter und übermittelt Benutzereingaben vom View an das Model. Die Aufteilung soll bei der Skalierbarkeit der Anwendung unterstützen und die Wiederverwendung von Komponenten ermöglichen. (Krasner, Pope et al., 1988)

Yang (2022) zeigt in dem Artikel auf, welche Anforderungen Web-Plattformen typischerweise erfüllen sollen und wie das vorgeschlagene Entwurfsmuster dabei helfen kann.

Der Journal Artikel von Prajwal et al. (2021) behandelt Micro Frontends und zeigt auf, wie diese den Gedanken von Microservices auf das Frontend übertragen. Der Artikel diskutiert mehrere technische Ansätze, um User Interfaces aus

Micro-frontends zusammenzusetzen. Zusätzlich gibt das Werk einen Überblick über Anwendungen aus der Praxis, die bereits Micro Frontends implementieren. Abschließend werden Vorteile und Nachteile von Micro Frontends dargestellt.

Peltonen et al. (2021) thematisieren ebenfalls Micro Frontends in der Publikation. Der Artikel gibt zuerst einen Überblick in die technische Umsetzung von Micro Frontends. Des Weiteren wurde eine strukturierte Literaturanalyse durchgeführt, welche Publikationen auf Motivationen, Vorteile und Probleme von Microfrontends untersucht. Die Ergebnisse werden aufbereitet und in einer Heat-Map dargestellt. Diese kann Anhaltspunkte für **(RQ2)** und **(RQ3)** liefern.

Ein- und Ausschlusskriterien

Nach Anwendung der Ein- und Ausschlusskriterien beschränkt sich die Anzahl relevanter Publikationen auf sechs. Werk No. **5** und Werk No. **6** werden als nicht geeignet eingestuft. Die Publikation von Fernández-Garcıa (2019) enthält keine englische Sprache. Lopes et al. (2018) präsentiert in der Dissertation eine Anwendung für Nicht-Entwickler, um Data Science Experimente durchzuführen. Die Anwendung ist cloud-basiert und baut auf einer Microservice-Architektur auf. Der grundsätzliche Fokus des Werks liegt aber nicht auf dem Thema Frontend.

C Case Study

Interview 1 Datum: 08.07.2022

Ort: Virtuelles Interview über Microsoft Teams

Interview 2 Datum: 06.07.2022

Ort: Virtuelles Interview über Microsoft Teams

C.1 Case Study Protokoll

Das Case Study Protokoll orientiert sich nach den von Pervan und Maimbo (2005) definierten Richtlinien. Tabelle 1 führt das Case Study Protokoll auf.

C.2 Interview Guide

I. Warm-up
1. Bitte stellen Sie sich und Ihre Rolle im Projekt vor.

II. Überblick System/Kontext
2. Bitte erläutern Sie kurz den Projekt-/Anwendungskontext (Thema, Team, Dauer)
3. Wie sieht das Nutzerspektrum der Anwendung aus? (wer, wie viele)
4. Wie ist das Projekt/die Anwendung organisatorisch aufgebaut?

III. Vertiefende Fragen
In unserer bisherigen Arbeit haben wir spezifischere Fragen identifiziert, denen wir nachgehen wollen. Wenn die Fragen nicht auf Sie zutreffen, teilen Sie uns dies bitte mit
5. Welchen architektonischen Stil verwenden Sie in Bezug auf die Frontend-Integration in Ihrem Projekt?
6. Haben Sie jemals die Frontend-Architektur im aktuellen Projekt geändert?
7. Warum haben Sie diese Entscheidung getroffen?
8. Bezogen auf Qualitätskriterien einer Anwendung (Performance, Robustheit, Usability und mehr) - haben Sie Empfehlungen für andere Praktiker bei der Auswahl der geeigneten Frontend-Archtitektur?
9. Wie kommuniziert das Frontend mit dem Backend auf technischer Ebene und umgekehrt?
10. Gibt es eine Kommunikation zwischen den Frontends? Wie wird diese konzipiert und umgesetzt?

IV. Herausforderungen
Aktuelle Herausforderungen
11. Was sind Ihre aktuellen (architektonischen, technischen, betrieblichen oder organisatorischen) Herausforderungen bei der Frontend-Integration in Microservice-Projekten?
12. Wie schaffen Sie es diese Herausforderungen zu bewältigen?

Vergangene Herausforderungen
13. Welche Herausforderungen mussten Sie in der Vergangenheit meistern?
14. Wie haben Sie diese überwunden?

V. Cool-down
15. In welchem Microservice Thema sehen Sie Verbesserungsbedarf bzw. Verbesserungspotenzial (allgemein)?

Sektion	**Inhalte**
Präambel	• Zweck des Case Study Protokolls: Verbesserung der Verlässlichkeit und Validität der Case Study • Anonymisierung personenbezogener Daten
Allgemeines	• Generierung von Forschungsdaten über User Interfaces in Microservice-basierten Systemen aus der Industrie • Forschungsdaten werden über die Forschungsfragen (RQ1 - RQ5) erzeugt • Interview Guide orientiert sich nach den Forschungsfragen RQ1 - RQ5 • Durchführung semi-strukturierter Interviews
Prozeduren	• Anzahl Cases: 2 • Ort: Virtuelles Interview über Microsoft Teams • Dauer der Session: 60 Minuten je Case • Aufnahme und Transkript des durchgeführten Interviews
Forschungsinstrumente	• Interview Guide, in Sektion C.2 beschrieben
Richtlinien zur Datenanalyse	• Triangulation: Durch die Auswahl von Personen in unterschiedlichen Rollen • Datenanalyse wie in strukturierter Literaturanalyse durchgeführt • Tabellarischer Aufbau der Ergebnisse ermöglicht Rückverfolgbarkeit • Transkript der Interviews und Zusendung an Teilnehmer zur Verifikation der Daten

Tabelle 1: Case Study Protokoll nach (Pervan & Maimbo, 2005)

C.3 Detaillierte Auswertung (RQ1-RQ4)

No.	Kategorie	Strategie	Beschreibung	Quellen	SLR
I-V1$_{S1}$	Vorteil	Monolithische UI	Höhere Entwicklungseffizienz	Interview 2, L. 105-118	Nein
I-N1$_{S1}$	Nachteil	Monolithische UI	Größerer Integrationsaufwand	Interview 2, L. 227-230	Ja
I-N2$_{S1}$	Nachteil	Monolithische UI	Koordination	Interview 1, L. 254-257	Ja
I-N4$_{S2}$	Nachteil	Micro Frontend	Schwierigere Konsistenz der UI	Interview 2, L. 143-147	Ja
I-N5$_{S2}$	Nachteil	Micro Frontend	Schwierigere Fehlerbehandlung	Interview 2, L. 207-217	Nein
I-N6$_{S2}$	Nachteil	Micro Frontend	Aufwendigere Integration bestimmter Wartbarkeitsszenarien	Interview 1, L. 156-164	Nein
I-N7$_{S2}$	Nachteil	Micro Frontend	Performance	Interview 1, L. 231-240	Ja
I-N8$_{S2}$	Nachteil	Micro Frontend	Größere Netzwerklast	Interview 1, L. 241-245	Nein
I-N9$_{S2}$	Nachteil	Micro Frontend	Höhere Komplexität und Onboarding	Interview 1, L. 163-171	Ja
I-H1$_{S2}$	Herausforderung	Generell	API Gateway als Antipattern	Interview 2, L. 383-386	Ja

C.4 Detaillierte Auswertung Guidelines (RQ5)

No.	Kategorie	Beschreibung	Quellen	SLR
I-G1	Guideline	Micro Frontends und Backend-Schnitt	Interview 1, L. 247-251 Interview 2, L. 199-224	Nein
I-G2	Guideline	UX-Design vor Micro Frontend Integration planen	Interview 1, L. 506-522	Nein
I-G3	Guideline	Micro Frontends um Komponenten außerhalb der Domäne einzubinden	Interview 2, L. 130-135	Nein

Tabelle 2: Detaillierte Auswertung

Literaturverzeichnis

Atchison, L. (2016). *Architecting for Scale: High Availability for Your Growing Applications*. Ö'Reilly Media, Inc."

Avison, D. E., Lau, F., Myers, M. D. & Nielsen, P. A. (1999). Action research. *Communications of the ACM*, *42*(1), 94–97.

Bugl, D. (2017). *Learning Redux*. Packt Publishing Ltd.

Carter, N., Bryant-Lukosius, D., DiCenso, A., Jennifer, B. & Neville, J. A. (2014). The use of triangulation in qualitative research. *41*(5), 545.

Conway, M. E. (1968). How do committees invent. *Datamation*, *14*(4), 28–31.

Fedosejev, A. (2015). *React. js essentials*. Packt Publishing Ltd.

Fernandes, J. L., Lopes, I. C., Rodrigues, J. J. & Ullah, S. (2013). Performance evaluation of RESTful web services and AMQP protocol. *2013 Fifth international conference on ubiquitous and future networks (ICUFN)*, 810–815.

Fernández-Garcıa, A. J. (2019). A Recommender System for Smart User Interfaces using Machine Learning and Microservices. *II JORNADAS*, 34.

Gackenheimer, C. & Paul, A. (2015). *Introduction to React* (Bd. 52). Springer.

Gehani, N. (2022). *Want to develop great microservices? Reorganize your team* [aufgerufen am 28. Oktober 2022]. https://techbeacon.com/app-dev-testing/want-develop-great-microservices-reorganize-your-team

Harms, H., Rogowski, C. & Lo Iacono, L. (2017). Guidelines for adopting frontend architectures and patterns in microservices-based systems. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 902–907.

Hassenzahl, M. & Tractinsky, N. (2006). User experience-a research agenda. *Behaviour & information technology*, *25*(2), 91–97.

Hong, X. J., Yang, H. S. & Kim, Y. H. (2018). Performance analysis of RESTful API and RabbitMQ for microservice web application. *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 257–259.

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, *33*(2004), 1–26.

Krasner, G. E., Pope, S. T. et al. (1988). A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, *1*(3), 26–49.

Li, K., Ding, Y., Shen, D., Li, Q. & Zhen, Z. (2020). The Design and Research of Front-End Framework for Microservice Environment. *2020 International Conference on Computer Information and Big Data Applications (CIBDA)*, 124–127.

Lopes, B. L. A. et al. (2018). *Data Science for Non-Programmers: Orchestration of Microservices and Graphical User Interface* (Diss.). Universidade de Coimbra.

Magnoni, L. (2015). Modern messaging for distributed sytems. *Journal of Physics: Conference Series*, *608*(1), 012038.

Montesi, F. & Weber, J. (2016). Circuit breakers, discovery, and API gateways in microservices. *arXiv preprint arXiv:1609.05830*.

Newman, S. (2015). *Pattern: Backends For Frontends: Single-purpose Edge Services for UIs and external parties*. https://samnewman.io/patterns/architectural/bff/

Newman, S. (2021). *Building microservices*. Ö'Reilly Media, Inc."

Peltonen, S., Mezzalira, L. & Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review. *Information and Software Technology*, *136*, 106571.

Pervan, G. & Maimbo, M. (2005). Designing a case study protocol for application in IS research. *Proceedings of the ninth pacific asia conference on information systems*, 1281–1292.

Prajwal, Y., Parekh, J. V. & Shettar, R. (2021). A Brief Review of Micro-frontends. *United International Journal for Research and Technology*, *2*(8).

Rademacher, F., Sorgalla, J. & Sachweh, S. (2018). Challenges of domain-driven microservice design: A model-driven perspective. *IEEE Software*, *35*(3), 36–43.

Richardson, L. & Ruby, S. (2008). *RESTful web services*. Ö'Reilly Media, Inc."

Runeson, P. & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131–164.

Susman, G. I. & Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative science quarterly*, 582–603.

Taibi, D. & Mezzalira, L. (2022). Micro-Frontends: Principles, Implementations, and Pitfalls. *ACM SIGSOFT Software Engineering Notes*, *47*(4), 25–29.

Takahashi, A. (2016). A "User-Flow Description" Method for Usability Investigation. *International Conference on Human-Computer Interaction*, 155–160.

Thönes, J. (2015). Microservices. *IEEE software*, *32*(1), 116–116.

Tilak, P. Y., Yadav, V., Dharmendra, S. D. & Bolloju, N. (2020). A platform for enhancing application developer productivity using microservices and micro-frontends. *2020 IEEE-HYDCON*, 1–4.

Webpack [aufgerufen am 13. November 2022]. (2022). https://webpack.js.org/

Wickham, M. (2018). Lazy Loading Images. *Practical Android* (S. 47–84). Springer.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 1–10.

Wolff, E. (2018). *Microservices: Grundlagen flexibler Softwarearchitekturen*. dpunkt. verlag.

Yang, Y. (2022). Research and Implementation of Web Front End Development System Based on Microservice. *International Conference on Cognitive based Information Processing and Applications (CIPA 2021)*, 1047–1052.

Zammetti, F. (2020). *Modern Full-Stack Development: Using TypeScript, React, Node. js, Webpack, and Docker*. Springer.