



Masterthesis

Microfrontend-Architekturen in Portalapplikationen:
Evaluierung und prototypische Implementierung

Erstellt von:

Matthis Wieneke
Straße
Stadt

Prüfer:

Prof. Dr. Jan Stehr
Matthias Füller

Eingereicht am:
21. April 2022

Inhaltsverzeichnis

Abbildungsverzeichnis	VI
Tabellenverzeichnis	VIII
Listingverzeichnis	IX
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Vorgehensweise	2
2 Grundlagen zu Microfrontends und Portalapplikationen	3
2.1 Webentwicklung	3
2.1.1 Softwarearchitektur	4
2.1.2 Angular Frontend	7
2.2 Microservices	8
2.3 Microfrontend-Architektur	12
2.3.1 Gemeinsamkeiten und Abgrenzungen zu Microservices	14
2.3.2 Microfrontend-Architektur aus technischer Sicht	14
2.3.3 Microfrontend-Architektur aus organisatorischer Sicht	23
2.4 Portalapplikationen	26
2.4.1 Portalapplikationen durch Hyperlinks	27
2.4.2 Portalshell	28
2.5 Microfrontends	30
2.5.1 Server-side Composition	31
2.5.2 Client-side Composition	33
2.5.3 Iframe	33
2.5.4 Web Components	35
2.5.5 Webpack Module Federation	36
2.5.6 Auswirkungen der Entscheidungsvielfalt	39
3 Prototypisches Beispiel in Angular	42
3.1 Architekturübersicht	43
3.2 Portalapplikation	44
3.3 Eingebundene Microfrontends	46

3.4 Anforderungen an das System	48
4 Evaluierung verschiedener Arten der Einbindung von Microfrontends	49
4.1 Theoretische Grundlagen einer Nutzwertanalyse	49
4.2 Kriterien für verschiedene Arten der Einbindung	54
4.3 Gewichtung der Kriterien	59
4.4 Analyse verschiedener Integrationsansätze	61
4.4.1 Abgrenzung unpassender Ansätze	61
4.4.2 Iframe	62
4.4.3 Web Components	70
4.4.4 Module Federation mit Angular Modulen	76
4.4.5 Module Federation mit Web Components	84
4.5 Vergleich der Arten untereinander	91
4.6 Optimale Anwendungsszenarien je nach Art der Einbindung	92
5 Prototypische Implementierung	95
5.1 Anpassungsbedarf der Beispielapplikation	95
5.2 Implementierung der Microfrontends	96
5.3 Validierung der Kriterien	97
6 Schlussbetrachtung	100
6.1 Zusammenfassung	100
6.2 Ausblick	100
Anhang	102
Quellenverzeichnis	138
Ehrenwörtliche Erklärung	139

Abbildungsverzeichnis

Abbildung 1: Beispielhafte Architekturmerkmale	5
Abbildung 2: MVVM Pattern	6
Abbildung 3: Unterschiede monolithische Anwendung zu Microservices	9
Abbildung 4: Verschiedene voneinander unabhängige Microfrontends	12
Abbildung 5: Frontend Integration der einzelnen Microfrontends	13
Abbildung 6: Eine aus Microfrontends bestehende Webseite	13
Abbildung 7: Horizontale und vertikale Trennung von Microfrontends	15
Abbildung 8: Organisationsmöglichkeiten der Microfrontend Teams	24
Abbildung 9: Zusammenfassende Darstellung Microfrontends	26
Abbildung 10: Verlinkung anderer Webseiten	27
Abbildung 11: Server- und Client-side Composition	31
Abbildung 12: Notwendige Konfigurationen für Module Federation	37
Abbildung 13: Weiterleitung und Zusammensetzung von Microfrontends	40
Abbildung 14: Microfrontend-Architektur Entscheidungsbaum nach Geers . .	41
Abbildung 15: Beispielapplikation Microfrontend-Architektur	43
Abbildung 16: Beispielapplikation Solutionarchitektur	44
Abbildung 17: Übersicht der Elemente der Portalshell	45
Abbildung 18: Design Mockup Dashboard <i>Prototyp</i>	47
Abbildung 19: Allgemeines Ablaufschema einer Nutzwertanalyse	51
Abbildung 20: Befüllte Präferenzmatrix mit Ergebnissen dreier Teilnehmer .	52
Abbildung 21: Rechenschema einer Nutzwertanalyse	54
Abbildung 22: Ausgefüllte Paarvergleichsmatrix	60
Abbildung 23: Übertragene Datenmenge bei mehreren Bibliotheken	90
Abbildung 24: Gegenüberstellung Ergebnisse der Nutzwertanalyse	92
Abbildung 25: Entscheidungsbaum: Einbindung Microfrontend in Portalshell	94
Abbildung 26: Prototypische Implementierung des Dashboard <i>Prototyp</i> . . .	98
Abbildung 27: Technologien für API Entwickler, Stand 07/2020	118
Abbildung 28: Konkrete Solutionarchitektur Beispielapplikation	118
Abbildung 29: Vergleichbares Microfrontend für Evaluierung	119
Abbildung 30: Evaluierung Messung Iframe	119
Abbildung 31: Evaluierung Messung Web Component	120
Abbildung 32: Evaluierung Messung Module Federation	120
Abbildung 33: Evaluierung Messung Module Federation Web Components .	120

Abbildung 34: HTML eines eingebundenen Iframes	121
Abbildung 35: Populärste Web Frameworks 2021	121
Abbildung 36: Marktanteile der führenden Browser	122
Abbildung 37: Teilen von Daten bei Module Federation	122
Abbildung 38: Durchgeführte Messungen mit durchschnittlicher Datenmenge	123
Abbildung 39: Doppelte Datenmenge durch zweifach eingebundenes Iframe .	123
Abbildung 40: Eingebundenes Iframe mit abgekapselten Stylings	124
Abbildung 41: Sequenzdiagramm einer Zusammensetzung durch Nginx SSI .	124
Abbildung 42: Entscheidungsbaum Technologie Microfrontends	125
Abbildung 43: Laden der Wetter Web Component	125
Abbildung 44: Event Emitter zum Übertragen von Daten	126
Abbildung 45: Web Storage zum Teilen von Daten	126
Abbildung 46: Beispielhafte realisierte Lokalisierung bei Amazon	127
Abbildung 47: Zusammengefassste Präferenzmatrix mehrerer Teilnehmer . . .	127
Abbildung 48: Web Component in der eigenen Anwendung eingebunden . . .	128
Abbildung 49: Module Federation WC Beeinflussung Styles	128
Abbildung 50: Ladezeiten Module Federation WC bei 8 geteilten Bibliotheken	129
Abbildung 51: Horizontaler Wissensaustausch in vertikalen Teams	129
Abbildung 52: Schnittmengen von Frameworks bei Module Federation	130
Abbildung 53: Monolithisches Frontend vs. Microfrontend	130
Abbildung 54: Skalierung Web Component doppelte Einbindung	131
Abbildung 55: Histogramm der Datenmenge von NPM-Paketen	131
Abbildung 56: Übersicht der Elemente der Portalapplikation mit Beispielen .	132
Abbildung 57: Laborsetup der Messungen	132
Abbildung 58: Funktionsweise einer Portalapplikation	133
Abbildung 59: Microfrontend mit <i>moment.js</i> geteilt durch Module Federation	133

Tabellenverzeichnis

Tabelle 1: Zusammenfassung des Microfrontend Decision Framework	19
Tabelle 2: Arbeitsschritte einer Nutzwertanalyse	50
Tabelle 3: Vergleichskriterien für Arten der Einbindung von Microfrontends	59
Tabelle 4: Entwicklungsaufwand verschiedener Microfrontends	67
Tabelle 5: Übersicht Datenmenge & Renderingzeit von Einbindungsarten .	68
Tabelle 6: Zusammenhang Kriterium zu Anwendungsfall	93
Tabelle 7: Verwendete NPM-Pakete zum Vergleich der Datenmenge	134
Tabelle 8: Übersicht der Rollen der prototypischen Portalshell	134

Listingverzeichnis

Listing 1:	Exemplarische Einbindung einer Webseite über das Iframe-Element	34
Listing 2:	Exemplarische Konfiguration Module Federation Portalshell . . .	103
Listing 3:	Exemplarische Konfiguration Module Federation Microfrontend .	104
Listing 4:	Konfiguration eines Module Federation Microfrontends	105
Listing 5:	Konfiguration einer Module Federation Portalshell	106
Listing 6:	Einbindung eines Module Federation Microfrontend	107
Listing 7:	Konfiguration von Module Federation mit Web Components . . .	108
Listing 8:	Definition des customElements zur Einbindung als Web Component	109
Listing 9:	Laden einer Web Component durch Module Federation	110
Listing 10:	Einbindung einer Web Component durch Module Federation . . .	111
Listing 11:	Platzhalter für SSI	111
Listing 12:	Nutzung eines HTML Templates	112
Listing 13:	Einbindung eines Taschenrechner als Iframe	113
Listing 14:	Quellcode der IframeInjection-Component	113
Listing 15:	Komponente zur Einbindung von Web Components	114
Listing 16:	Quellcode der Weather-Component	115
Listing 17:	Einbindung des Module Federation Microfrontends	116
Listing 18:	Einbindung einer Content-Component als Web Component . . .	117

1 Einleitung

In der nachfolgenden Masterthesis zum Thema *Microfrontend-Architekturen in Portalapplikationen: Evaluierung und prototypische Implementierung* wird das Konzept einer Microfrontend-Architektur anhand einer Portalshell und mehreren eingebundenen Microfrontends erklärt.

1.1 Motivation

Microservices sind seit vielen Jahren in der Softwareentwicklung etabliert. Mittlerweile sind zwei Drittel der repräsentativen Unternehmen auf dem Weg, ihre Software basierend auf Microservice-Technologien zu cloud-native Lösungen umzubauen.¹

Ein vergleichbarer Trend ist im Bereich von Frontends zu beobachten. In den vergangenen Jahren sind Microfrontends vermehrt auf dem Technologieradar erschienen.² Microfrontends bieten ebenso wie die Microservices viel Potential, sind aber noch weniger verbreitet. Durch Microfrontend-Architekturen können komplexe Webapplikationen realisiert werden, welche zeitgleich skalierbar sowie flexibel sind und von autonomen, vertikalen Teams betreut werden. Welche Microfrontend-Architekturen für den jeweiligen Anwendungsfall in Frage kommen und welche Besonderheiten diese jeweils mit sich bringen ist aufgrund der großen Entscheidungsvielfalt nicht immer direkt ersichtlich.

Zum aktuellen Zeitpunkt planen 73% der Firmen alle ihre Softwaresysteme auf Software as a Service (SaaS)-Lösungen umzustellen.³ Unternehmen, welche im Business-To-Business (B2B) Geschäftsfeld tätig sind, können von diesem Trend profitieren und umfangreiche, aber zeitgleich auf den individuellen Anforderungen basierende Softwarelösungen für ganze Branchen anbieten.

Dafür würde beispielsweise die Erstellung einer mandantenfähigen Portalapplikation in Frage kommen, in welcher B2B-Kunden ihre benötigten Tools in Form von Microfrontends individuell buchen, konfigurieren und nutzen können.

¹Lijnendonk2021

²Thoughtworks2020

³Alves2021

1.2 Zielsetzung

Ziel dieser Masterthesis ist es, eine Entscheidungsgrundlage für Portalapplikationen zu schaffen, anhand derer die optimale Art der Einbindung für Microfrontends gefunden werden kann. Die Entscheidungsfindung muss aufgrund der verschiedenen Handlungsoptionen fundiert durchgeführt werden, damit die Ergebnisse anwendbar sind. Jedes Microfrontend soll optimal in die Portalapplikation integriert sein, auf Basis der individuellen Anforderungen der Portalapplikation, des Nutzers und des jeweiligen Microfrontends.

1.3 Vorgehensweise

Um das Ziel zu erreichen gliedert sich die Masterthesis in sechs Kapitel. Nach der Einleitung in Kapitel 1, welche Motivation, Zielsetzung und die Vorgehensweise beschreibt, werden in Kapitel 2 die Grundlagen für die Arbeit gelegt. Die Grundlagen gehen auf Microservices, Microfrontends und Portalapplikationen ein, welche den fachlichen Rahmen der Arbeit ausmachen.

Anschließend wird in Kapitel 3 das prototypische Beispiel skizziert, anhand dessen in Kapitel 4 die Evaluierung durchgeführt wird. Die Evaluierung beinhaltet zunächst die Erläuterung der theoretischen Grundlagen einer Nutzwertanalyse, welche dann anschließend in den Abschnitten 4.2 bis 4.5 anhand des prototypischen Beispiels durchgeführt wird.

Im letzten Abschnitt des vierten Kapitels (4.6) wird das Ergebnis der Nutzwertanalyse interpretiert und aufbereitet. In diesem Abschnitt wird ein Prozess definiert, welcher einen Leitfaden bei der Entscheidungsfindung zur Einbindung von Microfrontends in eine Portalshell bilden soll.

Das vorletzte Kapitel beinhaltet die prototypische Implementierung anhand der vorherigen Erkenntnisse. Dort soll der entwickelte Prozess für einige exemplarische Microfrontends angewendet werden. Anschließend werden die Ergebnisse bewertet.

Das letzte Kapitel enthält eine Schlussbetrachtung über die vorangegangene Masterthesis. Im Zuge dieser wird die Erfüllung der festgelegten Zielsetzung überprüft und eine Wertung vollzogen. Anschließend wird ein Ausblick gegeben, welche weiteren Aspekte rund um das ausgewählte Thema ebenfalls untersucht werden könnten.

2 Grundlagen zu Microfrontends und Portalapplikationen

In diesem Kapitel werden die fachlichen Grundlagen für die Masterthesis vorgestellt. Dafür werden zunächst die Grundlagen der Webentwicklung erklärt und anschließend das Konzept von Microservices erläutert. Microservices werden danach gegenüber Microfrontends abgegrenzt, welche dann anschließend mitsamt dem Architekturkonzept erklärt werden.

2.1 Webentwicklung

Für eine Webanwendung ist ein Webfrontend erforderlich, welches bei Bedarf mit einem Backend kommuniziert. Webfrontends werden nachfolgend grundlegend erklärt. Backends sind nicht relevanter Bestandteil der Masterthesis und werden deswegen nicht näher betrachtet. Die Softwarearchitektur ist als übergreifendes Konzept relevant und wird daher im kommenden Abschnitt 2.1.1 erläutert.

Webanwendungen bestehen im Frontend üblicherweise aus drei Webstandards: *HyperText Markup Language (HTML)*, *Cascading Style Sheets (CSS)* und *Javascript (JS)*. *HTML* ist eine Auszeichnungssprache, welche den strukturellen Aufbau sowie den dargestellten Inhalt bestimmt. *CSS* gestaltet die Inhalte und formatiert sie nach Bedarf. Das Verhalten eines Webfrontends kann durch *JS* angepasst werden. Dadurch kann Validierung, Verhalten und Kommunikation mit dem Backend realisiert werden.⁴

Die drei Webstandards liefern einzeln keine angenehm nutzbare Webseite. Erst in Kombination untereinander können komplexe Webanwendungen geschaffen werden. Die Benutzbarkeit einer Webseite kann durch die *User Experience (UX)* beschrieben werden. Das Aussehen und die Darstellung wird durch das *User Interface (UI)* bestimmt.

⁴Erni2020

2.1.1 Softwarearchitektur

In diesem Abschnitt werden die Grundlagen der Softwarearchitektur erklärt, welche für die Masterthesis relevant sind.

Definition Softwarearchitektur

Softwarearchitektur ganzheitlich zu erläutern ist komplex und würde den Rahmen der Thesis überschreiten. Softwarearchitektur kann als Blaupause oder Richtlinie einer Softwarelösung verstanden werden.⁵

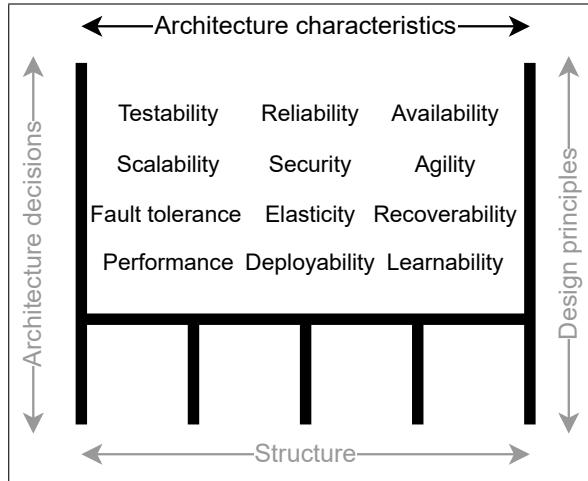
Eine mögliche Beschreibung von Softwarearchitektur sieht die Betrachtung von vier Seiten vor: *Architekturstruktur*, *Architekturmerkmale*, *Architekturentscheidungen* und *Designprinzipien*. Die *Architekturstruktur* einer Softwarelösung beschreibt, wie die Software vom Stil her aufgebaut ist. Beispielhaft wären eine Microservice-Architektur oder eine monolithische Architektur.⁶

Die *Architekturmerkmale* beschreiben die relevanten Anforderungen an das Gesamtsystem, welche auch zeitgleich die Funktionalität ausdrücken. Zur Verdeutlichung der Vielzahl und hohen Komplexität der Merkmale sind in der nachfolgenden Abb. 1 zwölf beispielhafte *Architekturmerkmale* einer Anwendung dargestellt. Beispielhaft wären *Verfügbarkeit*, *Sicherheit*, *Performance* und *Skalierbarkeit* als *Architekturmerkmale* einer Softwarelösung zu nennen. Die Merkmale lassen sich aus den Anforderungen an die Software ableiten.

⁵Ford2020

⁶Ford2020

Abbildung 1: Beispielhafte Architekturmerkmale



Quelle: Ford2020

Einige Merkmale schließen sich gegenseitig aus, wie es beispielsweise bei hoher *Konsistenz*, hoher *Verfügbarkeit* und hoher *Ausfalltoleranz* in einem verteilten System der Fall ist. Die Unmöglichkeit diese drei *Architekturmerkmale* gleichzeitig in einem verteilten System zu garantieren, wird als *CAP-Theorem* bezeichnet. Die Abkürzung CAP steht für die englischen Architekturmerkmale *Consistency*, *Availability* und *Partition tolerance*.⁷

Der dritte Faktor sind die *Architekturentscheidungen*, welche für eine Software relevant sind. Ein Softwarearchitekt gibt Regeln für das System vor, welche eingehalten werden müssen. Beispielsweise wird bei einer Schichtenarchitektur entschieden, dass nur die Business- und Serviceschichten die Datenbankschicht erreichen dürfen. Die Präsentationsschicht hingegen darf dies nicht. Diese Regeln setzen Grenzen für das System und weisen die Entwickler an, wie sie die Anwendung zu realisieren haben.⁸ Diese Regeln dürfen, im Gegensatz zu Best Practices, nicht verletzt werden.

Der letzte Aspekt der Softwarearchitektur sind *Designprinzipien*. Diese beinhalten im Gegensatz zu den strikten Regeln, welche bei den Architekturentscheidungen aufgestellt wurden, eher Empfehlungen und Best Practices, welchen gefolgt werden sollte. Ein Beispielhaftes Designprinzip wäre es, sofern möglich, immer auf asynchrone Kommunikation zu setzen.⁹

⁷IBM2019

⁸Ford2020

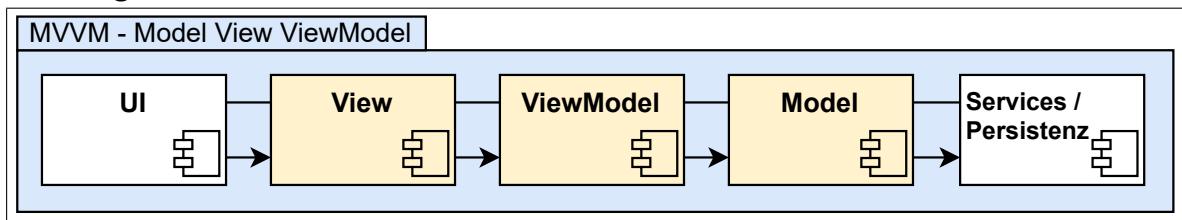
⁹Ford2020

Architekturstruktur MVVM

Eine mögliche Architekturstruktur für Webanwendungen ist das *Model-View-ViewModel (MVVM)* Entwurfsmuster, welches eine Variante des *Model-View-Controller (MVC)* Entwurfsmusters darstellt und in vielen Web Frameworks verwendet wird, wie bspw. *Angular*, *Vue.js* oder *React*.

In der nachfolgenden Abb. 2 ist das *MVVM* Pattern dargestellt, wie es beispielsweise bei Angular eingesetzt wird.

Abbildung 2: MVVM Pattern



Quelle: Tremp2021

Das *MVVM* Muster besteht aus drei zentralen Komponenten: *Datenmodell (Model)*, *Präsentation (View)* und *Programmsteuerung (ViewModel)*. Das *ViewModel* beinhaltet die Attribute der Geschäftsobjekte, welche von der *View* bei Bedarf konsumiert und angezeigt werden können. Das *Model* beinhaltet die Logik und kann, durch geänderte Daten, Änderungen an der *View* auslösen. Die *View* repräsentiert den Inhalt, der in dem UI angezeigt wird. Sie nimmt Nutzerinteraktionen entgegen und steht in bidirektionaler Kommunikation mit dem *ViewModel*, um Werte anzuzeigen oder Input weiter zu reichen.¹⁰

Designprinzip DDD

Bei dem Designprinzip des *Domain-Driven Design (DDD)* wird eine große Anwendung in kleine, nach Fachlichkeit getrennte, Domänen aufgeteilt. Die Domänen können anhand ihres Nutzens und der Nutzer, welche mit ihr interagieren, identifiziert werden. Eine Domäne ist selbstständig und abgekapselt gegenüber anderen Domänen. Komplexe Fachlichkeiten können in Sub-Domänen einer Domäne abgebildet werden.¹¹

¹⁰Tremp2021

¹¹Steyer2020

2.1.2 Angular Frontend

Im Jahr 2009 wurde mit *AngularJS* ein Web Framework für dynamische Webanwendungen veröffentlicht.¹² Angular Versionen sind nach *Semantic Versioning* benannt. *AngularJS* bezeichnet das Web Framework Angular in der Major Version 1. Seit der Major Version 2 wird das Web Framework generell als Angular bezeichnet.¹³ Zum aktuellen Zeitpunkt (Q1/2022) ist Angular 13, welches im November 2021 erschienen ist, die aktuellste Version.¹⁴

Durch das Web Framework Angular kann eine *Single Page Application (SPA)* realisiert werden. Bei einer *SPA* sind alle Inhalte auf einer einzigen Seite erreichbar, deren Bereiche dynamisch im Browser des Clients nachgeladen werden. Bei Angular wird die Steuerung der angezeigten Inhalte durch den Angular Router vorgenommen. Dieser kann Routen auch nur autorisierten Nutzern ermöglichen und somit Berechtigungslogik abbilden.¹⁵

Angular Anwendungen sind modular aufgebaut und in Module getrennt. Ein Angular Modul gruppiert Angular Komponenten und Services, welche zu einer Domäne gehören. Angular Module können exportiert werden. Dadurch können Module andere exportierte Module importieren, um deren Funktionen einzubinden.¹⁶

Eine Angular Komponente stellt eine View dar, welche aus Anzeige, Logik und Styling besteht und damit die drei Technologien verwendet, welche zu Beginn in Abschnitt 2.1 erläutert wurden.¹⁷ Die Darstellung und Änderung der Inhalte erfolgt über das MVVM Pattern. Eine Angular Komponente ist Teil eines Angular Modules und kann von anderen Komponenten in dem Modul referenziert und verwendet werden.

Damit Komponenten untereinander kommunizieren und auf gemeinsam genutzte Daten oder Logik zugreifen können, gibt es Angular Services. Durch Services wird außerdem die Logik, welche zu der View der Komponente gehört, von anderen Operationen getrennt.

¹²Clow2018

¹³Angular2022b

¹⁴Thompson2021

¹⁵Angular2022c

¹⁶Angular2022d

¹⁷Angular2022e

Dadurch beinhaltet die Komponente nur noch die eigene relevante Logik. Services können durch Konstruktoren Dependency Injection in der Komponente verfügbar gemacht werden und sind anschließend referenzierbar.¹⁸

Der Code von Angular Applikationen wird durch *webpack* als Modulbundler zu einer JS Datei zusammengefasst und anschließend zum Laden an den Client bereitgestellt. Webpack erstellt Pakete, welche die notwendigen HTML, CSS und JS Dateien enthalten. Ebenfalls werden die Referenzen zu eventuellen Assets gesetzt, welche ebenfalls mit veröffentlicht werden. Es werden die Import-Statements jeder Datei gelesen und anhand dieser eingebundenen Dependencies ein zusammenhängendes Bundle von benötigten Bibliotheken erstellt. Die relevante Konfiguration wird in der `webpack.config.js`-Datei definiert.¹⁹

2.2 Microservices

In diesem Abschnitt werden die Grundlagen zu Microservices erklärt. Microservices sind ein umfangreiches Thema, welches bei ganzheitlicher Betrachtung den Rahmen der Arbeit überschreiten würde. Daher werden nachfolgend nur die Grundlagen erklärt und anschließend Abgrenzungen und Gemeinsamkeiten zu Microfrontends erläutert.

Microservice-Architekturen zählen momentan zu den gefragtesten Technologien in der Softwareentwicklung²⁰ und sind nach der Lünendonk Studie 2021 „Cloud-native Software Development“ bei zwei dritteln der repräsentativen Unternehmen im Einsatz.²¹

Sam Newman definiert Microservices wie folgt: „Microservices are independently releasable services that are modeled around a business domain.“²²

In der nachfolgenden Abb. 3 ist der Unterschied zwischen einer monolithischen Anwendung und einer Microservice-Architektur mit einem monolithischen Frontend dargestellt.

¹⁸Angular2022f

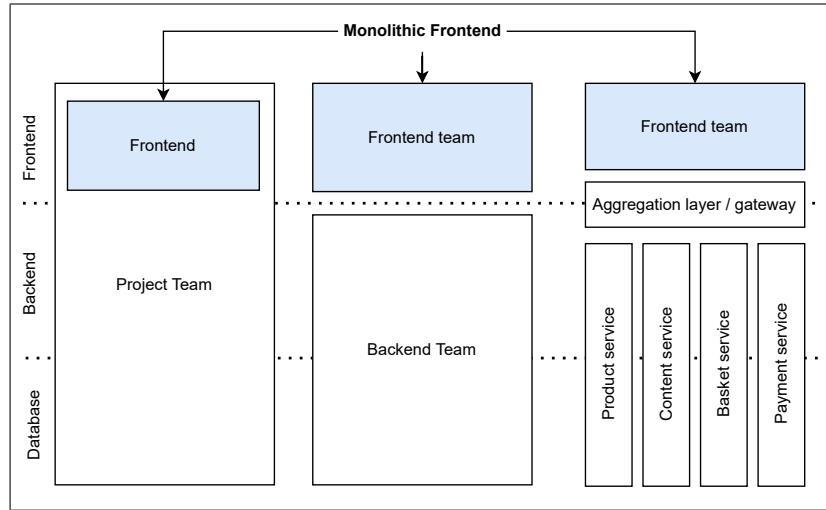
¹⁹Angular2022g

²⁰Siehe Abb. 27 in Anhang 2

²¹LÄijnendonk2021

²²Newman2021

Abbildung 3: Unterschiede monolithische Anwendung zu Microservices



Quelle: Geers2020

Während die monolithische Anwendung links in der Abbildung das Frontend und das Backend ganzheitlich und ohne Trennungen beinhaltet, kann alternativ das in sich monolithische Frontend von dem monolithischen Backend getrennt sein. Bei einer Microservice-Architektur ist das Frontend ebenfalls vom Backend getrennt, welches wiederum in einzelne funktionale Blöcke aufgeteilt ist, die sogenannten Microservices. Ein Microservice ist durch ein *Application Programming Interface (API)* aufrufbar und beinhaltet die Logik einer Domäne.

Ein Microservice kann durch die dedizierte Verantwortung einer Domäne unabhängig veröffentlicht, sowie eigenständig getestet und individuell skaliert werden.²³ Dies ist bei einer monolithischen Anwendung nicht gegeben. Generell können sieben Prinzipien von Microservices herausgestellt werden, durch die sie sich von monolithischen Lösungen abgrenzen.²⁴ Diese werden nachfolgend erläutert.

Modeled Around Business Domains

Nach dem Prinzip des DDD ist die Softwarelösung in fachliche Domänen aufgeteilt. Jeder Microservice repräsentiert eine eigene Domäne. Dies hat den Vorteil, dass das System klar strukturiert ist. Das verantwortliche Team der Domäne kennt die Grenzen der eigenen Verantwortung, da diese innerhalb ihrer Domäne liegt. Die Verantwortung anderer Domänen liegt bei anderen Teams.²⁵

²³Thoenes2015

²⁴Mezzalira2021

²⁵Mezzalira2021

Culture of Automation

Eine hoher Automatisierungsgrad ist bei einer Microservice-Architektur essentiell. Weil die Microservices unabhängig voneinander veröffentlicht werden sollen, empfiehlt es sich alle benötigten Schritte im *Continuous Integration, Continuous Delivery und Continuous Deployment (CI/CD)* Prozess zu automatisieren.²⁶ So kann das Team, welches bspw. für die Werbung verantwortlich ist, seinen Microservice automatisiert neu veröffentlichen. Es muss sich nicht mit den anderen Teams absprechen, solange keine breaking Changes veröffentlicht werden. Es ist nur der eigene Microservice kurzfristig von einem Ausfall während des Veröffentlichungsprozesses betroffen, was für eine starke Unabhängigkeit spricht.

Hide Implementation Details

Um Abhängigkeiten zu vermeiden werden alle nicht benötigten Features und Schnittstellen gegenüber anderen Domänen versteckt. Es werden nur benötigte Funktionen über APIs bereitgestellt. Dadurch behält das Team des Microservices den Überblick, welche Abhängigkeiten zu anderen Domänen bestehen und kann Änderungen bewusst durchführen, ohne bestehende andere Microservices zu beeinflussen.²⁷

Decentralize Governance

Bei monolithischen Anwendungen werden Architekturentscheidungen ganzheitlich für das gesamte System gefällt, obwohl diese Entscheidung eventuell nicht für jede Domäne optimal ist. Bei Microservices kann jedes Team seine eigenen Entscheidungen fällen. Ob sie beispielsweise synchron oder asynchron kommunizieren und welche Datenbanktechnologie eingesetzt wird, ist jedem Team selber überlassen. Dies sorgt dafür, dass in jeder Domäne die optimale Lösung für den jeweiligen Anwendungsfall zu finden ist. Dennoch müssen übergreifende Entscheidungen und Rahmenbedingungen vorgegeben sowie Standards eingehalten werden.²⁸

Deploy Independently

Individuelles Deployment ist durch eine hohe Automatisierungsrate zeitsparend realisierbar. Wenn jeder Microservice eine eigene Deploymentpipeline hat, können einzelne Domänen bei Bedarf in einer neueren Version veröffentlicht werden, ohne das anderen Domänen beeinträchtigt werden. Bei einer monolithischen Anwendung muss das gesamte System zusammen veröffentlicht werden, was zu einer Ausfallzeit führt und

²⁶ Mezzalira 2021

²⁷ Mezzalira 2021

²⁸ Mezzalira 2021

in Summe länger dauert. Microservices können einzeln schneller veröffentlicht werden und bei Bedarf ebenfalls schnell wieder auf einen funktionierenden Stand zurückgerollt werden.²⁹

Das unabhängige Veröffentlichen und ein hoher Automatisierungsgrad im Prozess der Veröffentlichung haben Synergieeffekte, weil der Releaseprozess dadurch pro Durchführung weniger Zeitaufwand benötigt.

Isolate Failure

Eine Folge der Auf trennung des monolithischen Backends in einzelne Microservices ist eine erhöhte Ausfallsicherheit. Bei einer monolithischen Anwendung ist das ganze System von einem Ausfall betroffen, bei einer Microservice-Architektur lediglich die Features der betroffenen Domäne. Kunden können weiterhin die Features der anderen Domänen nutzen und bemerken den Ausfall der betroffenen Domäne im besten Fall gar nicht.³⁰

Highly Observable

Eine der Herausforderungen einer verteilten Microservice-Architektur ist es, den Überblick über alle Domänen zu behalten. Eine monolithische Anwendung als Gesamtsystem ist einfacher zu Überblicken und kann schneller um neue domänenübergreifende Anfragen erweitert werden. Bei einer Microservice-Architektur ist dies mit mehr Aufwand verbunden, welcher aber durch die gewonnene Flexibilität zu rechtfertigen ist.³¹ Der höhere Aufwand entsteht durch die Abkapselung der anderen Domänen. Es ist für ein Team einer Domäne nicht direkt ersichtlich, von welchen Features anderer Domänen profitiert werden kann.

Zusammenfassend bringt eine Microservice-Architektur einige Vorteile gegenüber einer monolithischen Architektur. Allerdings ist eine Umstrukturierung der internen Organisation erforderlich, damit Teams unabhängig voneinander sind und Domänenentrennung bei Erhaltung des Gesamtüberblickes realisiert werden können.³²

Die Prinzipien von Microservices lassen sich auf Microfrontends übertragen, was im nachfolgenden Abschnitt 2.3 getan wird.

²⁹ Mezzalira 2021

³⁰ Mezzalira 2021

³¹ Mezzalira 2021

³² Mezzalira 2021

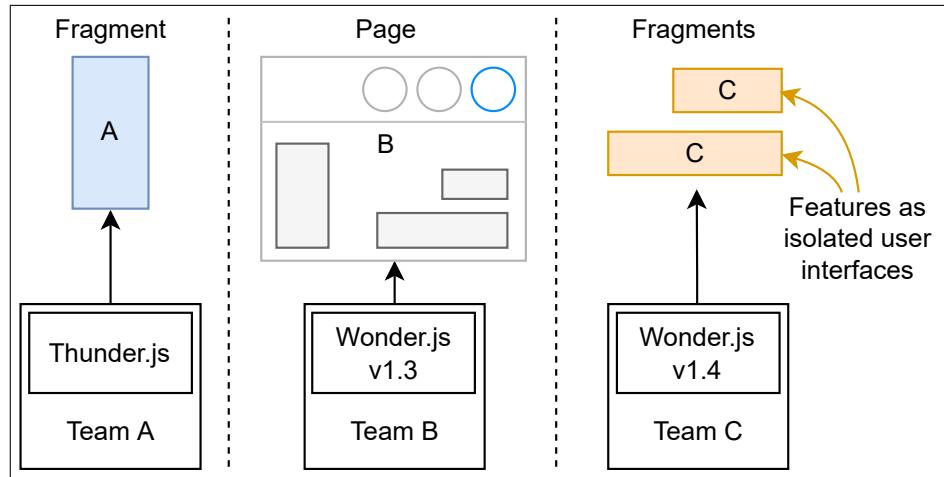
2.3 Microfrontend-Architektur

In diesem Kapitel werden die Grundlagen von Microfrontends erläutert. Dazu werden zunächst die Gemeinsamkeiten von Microfrontends gegenüber Microservices erläutert sowie Unterschiede abgegrenzt. Anschließend wird das Konzept einer Microfrontend-Architektur präsentiert und Microfrontends aus technischer sowie organisatorischer Sicht vorgestellt.

Microfrontends sind ein möglicher Ansatz der Softwarearchitektur einer Webseite. Anstatt eines monolithischen Frontends ist das Frontend in kleinere, fokussierte Subsysteme aufgeteilt, welche von einem eigenen Team betrieben werden, das sich auch um das dazugehörige Backend kümmert.³³

In der nachfolgenden Abb. 4 sind drei Microfrontends dargestellt. Diese werden jeweils von drei verschiedenen Teams betreut und können in unterschiedlichen Web-Frameworks entstanden sein.

Abbildung 4: Verschiedene voneinander unabhängige Microfrontends

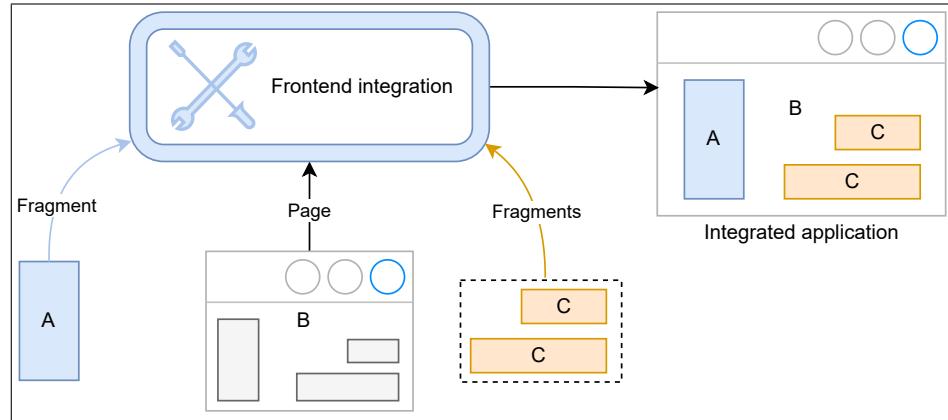


Quelle: Geers2020

Team A und C steuern jeweils Fragmente bei. Team B ist verantwortlich für die eigentliche Seite, welche die anderen Fragmente später einbindet, und lässt Platzhalter an den Stellen frei, wo die Fragmente in Zukunft platziert werden sollen. Das Zusammenfügen der einzelnen Fragmente auf der Seite erfolgt durch die Konfiguration von Team B und ist nachfolgend in der Abb. 5 dargestellt.

³³Geers2020

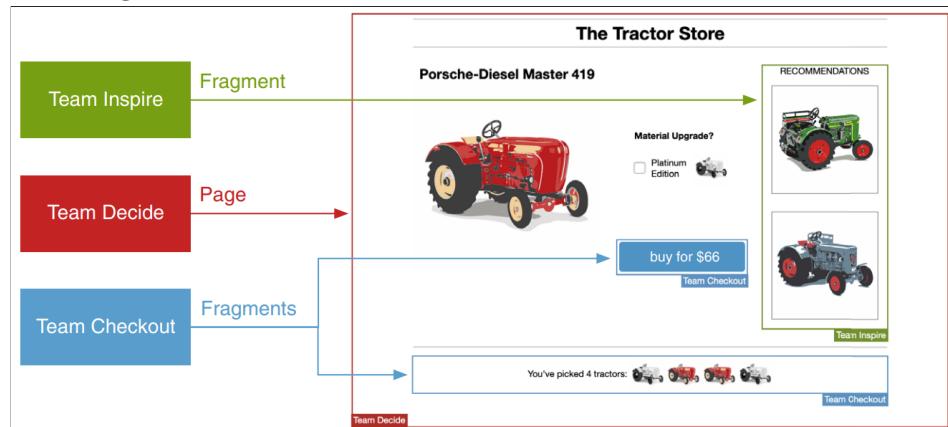
Abbildung 5: Frontend Integration der einzelnen Microfrontends



Quelle: Geers2020

Die Fragmente und die Seite mit den noch leeren Platzhaltern werden durch die sogenannte *Frontend integration* zusammengesetzt und es entsteht eine Web-Anwendung bestehend aus mehreren Microfrontends, was für den Anwender idealerweise nicht ersichtlich ist. Ein Beispiel für eine zusammengesetzte Microfrontend-Applikation ist in der nachfolgenden Abb. 6 dargestellt.

Abbildung 6: Eine aus Microfrontends bestehende Webseite



Quelle: Geers2020

Das Frontend des beispielhaften Traktorshops ist aus den Fragmenten drei verschiedener Teams zusammengesetzt. *Team Decide* stellt die eigentliche Seite mit den Platzhaltern zur Verfügung, in welche die Fragmente der Teams *Inspire* und *Checkout* eingebunden werden. Für den Shopbenutzer ist diese Trennung in drei Teams nicht ersichtlich und die Webseite erscheint als eine zusammengehörige Komponente.

2.3.1 Gemeinsamkeiten und Abgrenzungen zu Microservices

Im Abschnitt 2.2 wurden das Konzept, die Vorteile und Anwendungsfälle von Microservices vorgestellt. In diesem Abschnitt werden weiterführend Abgrenzungen und Gemeinsamkeiten von Microservices gegenüber Microfrontends herausgearbeitet, bevor im nachfolgenden Abschnitt 2.3.2 die Microfrontend-Architektur erklärt wird.

Microfrontends und Microservices unterscheiden sich durch ihren Scope. Microservices sind für das Backend einer Anwendung zuständig und befinden sich hinter der API. Microfrontends sind für das Frontend einer Anwendung zuständig und rufen die API auf.

Abseits vom Scope gibt es aber viele Gemeinsamkeiten. So lassen sich die Prinzipien von Microservices, welche im vorherigen Abschnitt dargestellt wurden, auf Microfrontends übertragen.³⁴ In Anhang 2 in Abb. 53 ist eine Gegenüberstellung von einem monolithischen Frontend zu einer Applikation mit Microfrontends zu sehen. Bei der Applikation mit Microfrontends ist das Frontend in mehrere unabhängige und nach Fachlichkeit getrennte Teile aufgetrennt, genau wie es bei Microservices im Backend der Fall ist. Dadurch sind vertikale Teams in der Lage die gesamte Verantwortung in ihrer Domäne zu übernehmen. Beispielsweise ist das *Product Team* für den *Product Microservice* sowie für das *Product Microfrontend* zuständig und verwaltet somit die gesamte Produkt-Domäne.

2.3.2 Microfrontend-Architektur aus technischer Sicht

„Microfrontends are not appropriate for every application because of their nature and the potential complexity they add at the technical and organizational levels.“³⁵

Damit die Vorteile von Microfrontends genutzt werden können, sollten die Anforderungen geprüft und die Architektur dahingehend geplant werden. Eine Microfrontend-Architektur kann nach Mezzalira durch vier Prinzipien beschrieben werden.

Die vier Prinzipien lauten: *business domain representation*, *autonomous codebase*, *independent deployment* und *single-team ownership*. Ein Microfrontend repräsentiert demen-

³⁴Mezzalira2021

³⁵Mezzalira2021

sprechend eine Geschäftsdomäne, welche autonom entwickelt sowie unabhängig veröffentlicht werden kann und von einem eigenen Team betreut wird.³⁶

Nachfolgend wird das sogenannte *Microfrontend Decision Framework* vorgestellt. Es beschreibt Kriterien, anhand welcher passende Architekturentscheidungen für die Anwendung getroffen werden können und beinhaltet vier Kernaspekte:³⁷

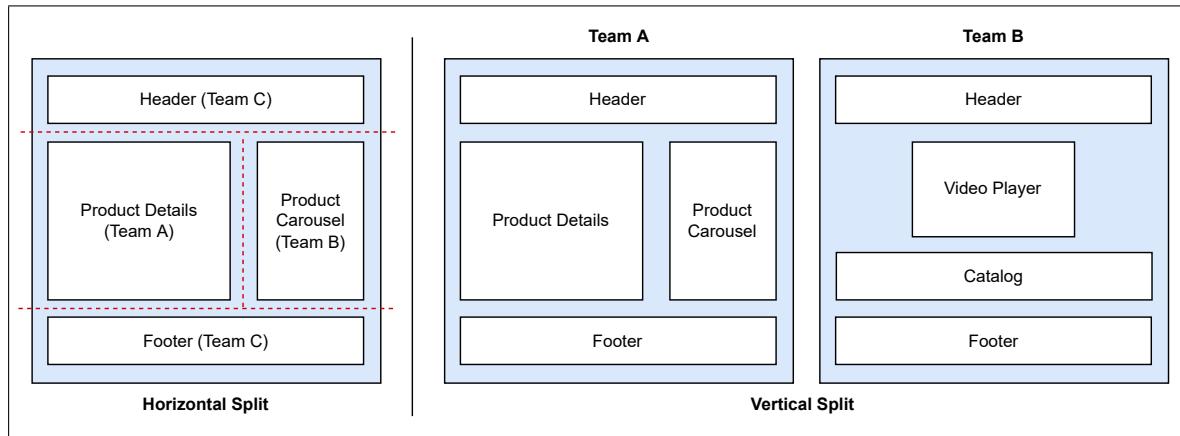
- Einheitliche Definitionen in der eigenen Architektur
- Zusammensetzung
- Weiterleitung
- Kommunikation untereinander

Die einzelnen Aspekte werden nachfolgend im Detail erklärt, weil sie wichtige Grundlagen zum Verständnis von Microfrontend-Architekturen bilden.

Einheitliche Definition von Microfrontends in der eigenen Architektur

Zunächst muss ein einheitliches Verständnis von einem Microfrontend gewonnen werden. Dafür wird unterschieden, ob es sich um eine vertikale Trennung in Domänen nach dem DDD Konzept, oder um eine horizontale Trennung von mehreren Microfrontends auf der gleichen Seite, wie in der nachfolgenden Abb. 7 dargestellt, handelt.

Abbildung 7: Horizontale und vertikale Trennung von Microfrontends



Quelle: Mezzalira2021

Bei dem DDD geht es darum, die Anwendung in Fachdomänen aufzutrennen. Eine Domäne kann wiederum in mehrere Subdomänen aufgeteilt werden, welche sich in drei

³⁶Mezzalira2021

³⁷Mezzalira2021

Typen unterscheiden lassen:

In *funktionale Subdomänen*, welche den Hauptnutzen der Domäne repräsentieren.

In *unterstützende Subdomänen*, welche zwar die funktionale Subdomäne unterstützen, aber keinen direkten Wert beisteuern.

In *generische Subdomänen*, welche der übergreifenden Plattform dienen und den Prozess abrunden. Sie haben keinen direkten Bezug zur Domäne, sind aber für die Erfüllung des übergreifenden Nutzens relevant (bspw. die Authentifizierung auf einer Webseite).³⁸

Im Falle der in Abb. 7 dargestellten vertikalen Trennung wird in die Domänen *Video* und *Produkt* unterschieden, welche jeweils von einem Team betreut werden. Die Teams mit ihren Domänen sind voneinander unabhängig und abgekapselt. Sollten Zugriffe domänenübergreifend notwendig sein, werden diese über ein API realisiert.

Zusammensetzung von Microfrontends

Ebenfalls muss vorab entschieden werden, wann die Microfrontends zusammengesetzt werden (engl. *composition*). Dies kann an drei verschiedenen Stellen geschehen:³⁹

- *Client-side Composition*
- *Edge-side Composition*
- *Server-side Composition*

Bei der *Client-side Composition* werden die einzelnen Microfrontends vom Client einzeln aus einem Content Delivery Network (CDN), oder falls dort nicht vorhanden, direkt aus der Quelle geladen und am Client im Browser zusammengesetzt.

Bei der *Edge-side Composition* wird das Zusammensetzen der einzelnen Microfrontends im CDN vollzogen und die Anwendung als Ganzes an den Client ausgeliefert.

Die *Server-side Composition* hingegen setzt die Microfrontends bereits auf einem Server zusammen und gibt sie als Ganzes an das CDN weiter, welches die Seite zwischenspeichert und anschließend an den Client ausliefert.

Client-side und *Server-side Composition* werden in den Abschnitten 2.5.1 bis 2.5.2 ausführlicher behandelt. Auf eine detailliertere Erklärung von *Edge-side Composition* wird verzichtet, da es dem Konzept der *Server-side Composition* ähnelt und für den weiteren Verlauf der Masterthesis keine Relevanz hat.

³⁸Mezzalira2021

³⁹Mezzalira2021

Weiterleitung von Microfrontends

Nachdem der Ort der Zusammenstellung der Microfrontends bestimmt wurde, muss eine Technik gewählt werden, um zwischen den Ansichten verschiedener Microfrontends zu wechseln. Bezogen auf den vorherigen Punkt *Zusammensetzung* muss entschieden werden, ob das Weiterleiten (engl. *Routing*) auf dem Server, dem CDN oder am Client passiert.

Wenn die Anwendung auf dem Server zusammengesetzt wird, muss sie auch dort gerouted werden, weil sich die Logik sowie die anderen Ansichten serverseitig befinden. Die gerouteten Ansichten können dann vom CDN zwischengespeichert werden, wenn sich die Ansichten nicht individuell pro Mandant oder User unterscheiden.⁴⁰

Bei dem Einsatz von *Edge-side Composition* wird das Routing über Uniform Resource Locator (URL) realisiert und die Microfrontends werden vom CDN Server zusammengezetzt.⁴¹ Es besteht in diesem Fall nicht viel Entscheidungsfreiraum beim Routing, was einer der Gründe ist, warum *Edge-side Composition* im Rahmen der Thesis nicht weiter verfolgt wird.

Das Routing am Client durchzuführen bringt die meiste Flexibilität, da die einzelnen Microfrontends nach Bedarf und individueller Konfiguration des Clients geladen und zwischengespeichert werden können.

Die Art des clientseitigen Routings ist in SPAs wiederzufinden, welche bereits vorher in Abschnitt 2.1.2 beschrieben wurden. Eine SPA kann als *Application Shell* fungieren, indem sie abhängig des aktuellen Router- und Nutzerstatus andere Microfrontends einbindet und anzeigt. Die *Application Shells* bieten Querschnittsaspekte, wie bspw. Routing, Authentifizierung und Autorisierung, von welchen die Microfrontends profitieren können und über die ein komplexes, individuelles Routing abgebildet werden kann. Die Konzepte der *Application Shell* und der eingebundenen Microfrontends werden im nachfolgenden Abschnitt 2.4 ausführlich behandelt.

Das clientseitige Routing kann auch ohne eine SPA lediglich über URLs realisiert werden, zwischen welchen der Benutzer navigieren kann.⁴² Dabei springt der Benutzer aber von Seite zu Seite, verliert eventuell gespeicherte Informationen und wird mit Ladezeiten je Seite konfrontiert.

⁴⁰ Mezzalira 2021

⁴¹ Mezzalira 2021

⁴² Mezzalira 2021

Kommunikation von Microfrontends untereinander

Optimalerweise laufen Microfrontends vollkommen autark und haben nach dem DDD Prinzip keine Überschneidungen zu anderen Domänen. Dies ist bei größeren Applikationen aber nicht gegeben, weil beispielsweise Microfrontends auf die Nutzerdomäne zugreifen müssen, um Informationen über den aktuellen Benutzer erhalten zu können.

Besonders bei mehreren Microfrontends auf der gleichen Seite ist das Verwalten einer konsistenten und kohärenten UX herausfordernd. Zusätzlich erschwert die Domänenverwaltung durch verschiedene Teams das Kommunizieren von Microfrontends untereinander.⁴³

Die Kommunikation kann, bei horizontaler Trennung, über vier verschiedene Arten erfolgen: *Event emitter*, *Custom events*, *Web Storage* oder *Query Strings*. Bei vertikaler Trennung sind es nur über einen *Web Storage* und *Query Strings* möglich.

Sind die Microfrontends auf der gleichen Seite eingebunden, kann über Events kommuniziert werden. Eine zentrale Stelle (bspw. die *Application Shell*) versendet Events und Microfrontends können auf diese Eventbenachrichtigungen reagieren. Dies ist über das *CustomEvent* Interface von Javascript möglich.⁴⁴

Die Daten können ebenfalls über *Event Emitter* geteilt werden, wie in Abb. 44 in Anhang 2 dargestellt. Dafür muss ein Eventbus in jedes Microfrontend eingefügt werden, auf welchem Nachrichten versendet werden können.⁴⁵

Eine weitere Möglichkeit zum Datenaustausch ist das Teilen von Informationen über einen gemeinsamen *Web Storage* (siehe Abb. 45 in Anhang 2). Es wird ein *Web Storage* verwendet, welcher von Microfrontends der gleichen Subdomain erreicht werden kann. Dort liegen Informationen des Benutzers, auf welche das Microfrontend zurückgreifen kann.⁴⁶

Zusammenfassung Microfrontend Decision Frameworks

In der nachfolgenden Tabelle 1 ist eine Zusammenfassung des Microfrontend Decision Frameworks dargestellt.

⁴³ Mezzalira 2021

⁴⁴ MDN Web Docs 2022b

⁴⁵ Mezzalira 2021

⁴⁶ Mezzalira 2021

Tabelle 1: Zusammenfassung des Microfrontend Decision Framework

Microfrontend definition	Composition	Routing	Communication
Horizontal	Client side	Client side	Event emitter
	Server side	Server side	Custom events
	Edge side	Edge side	Web storage Query strings
Vertical	Client side	Client side	Web Storage
	Server side	Server side	Query Strings
		Edge side	

Quelle: Mezzalira2021

Die Tabelle zeigt, dass nicht alle Ausprägungen einer Kategorie miteinander kombinierbar sind. Die erste Entscheidung muss bezüglich der Definition getroffen werden. Eine vertikale Definition hat zur Folge, dass beim Routing und bei der Kommunikation weniger Optionen verfügbar sind.

Die Vorteile und Nachteile lassen sich aus den eingangs erklärten Prinzipien von Microfrontends ableiten und werden zur Übersicht nachfolgend noch einmal erklärt.

Vorteile gegenüber einem monolithischen Frontend

Der primäre Vorteil einer Microfrontend-Architektur ist die gewonnene Flexibilität. Jedes Team ist selbstverantwortlich für ihr zugewiesenes Microfrontend. Sie sind in der Lage selbstständig Architekturentscheidungen zu treffen, welche zu ihren Aufgaben und Problemen passen. Ebenfalls können bei Bedarf neue Softwarestände durch die eigenen Pipelines veröffentlicht werden, ohne andere Microfrontends zu beeinflussen.

Des Weiteren wird das Microfrontend nicht durch andere Ausfälle beeinflusst. Hat beispielsweise das für die Werbung einer Seite verantwortliche Team eine Störung, kann das Team verantwortlich für die Kaufabwicklung ohne Auswirkungen weiter agieren.⁴⁷ So ist das nicht direkt betroffene Team weiterhin nicht abhängig und der Benutzer bemerkt die Störung außerhalb der betroffenen Domäne ebenfalls nicht.

Darüber hinaus ist durch die Trennung nach Domänen in eigenverantwortliche Teams eine klare Strukturierung der Verantwortung gegeben, welche bei der Organisation der Gesamtwebseite von Vorteil ist. Auch ist eine verbesserte Skalierbarkeit gegeben, sodass

⁴⁷Mezzalira2021

modulare Microfrontends mehrfach nach Bedarf eingebunden werden können (*Don't repeat yourself (DRY)* Prinzip).

Nachteile gegenüber einem monolithischen Frontend

Nachteile einer Microfrontend-Architektur gegenüber einem monolithischen Ansatz sind in den Aspekten Redundanz, Konsistenz und Heterogenität zu finden.⁴⁸

Redundanz

In Microfrontend-Architekturen sind Redundanzen nicht zu vermeiden. So hat jedes Team ein eigenes Repository, eine eigene Deployment-Pipeline und einen eigenen Ort zum Hosten des Microfrontends.⁴⁹ Ein eigenes Repository ist zwar hilfreich, um sich als Team selbstständiger zu organisieren, aber es besteht auch die Möglichkeit ein Monorepo übergreifend zu verwenden. Dort sind alle Microfrontends der Applikation gleichzeitig enthalten. Alle Microfrontends können auf geteilte Bibliotheken in dem Monorepo zugreifen. Dies erfordert eine gute Zusammenarbeit der einzelnen Teams und eine strikte Organisation des Repositories, was mit Mehraufwand einhergeht.⁵⁰

Durch den Aspekt, dass das vorher monolithische Frontend in kleinere Fragmente aufgeteilt wurde, welche alle selbstständig lauffähig sind, kommt es zu mehrfach geladenem Code.⁵¹ Beispielsweise muss das Web Frontend Framework Angular für jedes Microfrontend geladen werden oder es müssen mehrere verschiedene Web Frontend Frameworks geladen werden. Bei einer monolithischen Anwendung dagegen wird nur ein Web Frontend Framework benötigt.

Konsistenz

Der Microfrontend Ansatz verlangt, wie in Abb. 9 zu sehen, die Aufteilung in eigenständige, vertikal agierende Teams. Diese Teams sollen nach Möglichkeit alles selbstständig betreiben, vom Frontend über das Backend bis hin zur Datenbank. Nun kann es vorkommen, dass ein Team Daten von einem anderen Team benötigt. Ein direkter Zugriff oder eine geteilte Datenbank würde die Domänenentrennung verletzen.

Aus diesem Grund muss das Team, welches Daten eines anderen Teams benötigt, die Daten replizieren. Das Replizieren von Datenbeständen ist notwendig, damit aus

⁴⁸Geers2020

⁴⁹Geers2020

⁵⁰Mezzalira2021

⁵¹Geers2020

Gründen der Unabhängigkeit und Ausfallsicherheit auf direkte Zugriffe verzichtet werden kann. Die Replizierung der Daten hat den Nachteil, dass Abhängig von den Replizierungsintervallen unter Umständen veraltete Datenbestände verwendet werden. Darüber hinaus werden doppelte Ressourcen (Speicher- und Rechenkapazität) benötigt.⁵²

Auf eine Replizierung kann verzichtet werden, wenn die Schnittstellen von Domänen in begründeten Fällen doch zur Verfügung gestellt werden. Dies könnte beispielsweise bei weniger relevanten Daten der Fall sein, bei denen ein temporärer Ausfall die Kernfunktion nicht beeinträchtigt.

Heterogenität

Da die Teams unabhängig voneinander sind, steht es ihnen frei, welche Technologie sie verwenden. So können alle Microfrontends einer Applikation einheitlich in Angular 12 geschrieben sein. Es besteht aber auch die Möglichkeit verschiedene Versionen der gleichen Technologie zu verwenden. Ein Angular 11 Microfrontend kann in eine Angular 12 Application Shell eingebunden werden. Es können aber auch andere Web Frontend Frameworks verwendet werden, also an Stelle von Angular beispielsweise Vue.js oder React. Allerdings erschweren heterogene Technologien den Informationsaustausch der Teams untereinander und einzelne Entwickler können nicht so einfach ohne Weiterbildung zwischen den Teams wechseln.

Einheitliches Design ist ein wichtiger Aspekt einer funktionierenden Software. Es gibt dem User das Gefühl sich durchgehend auf einer Webseite aufzuhalten. Durch ein *Design System*, eine verbindliche Vorgabe zur Einhaltung eines stringenten Designs für alle beteiligten Teams der Applikation, kann eine einheitliche Webseite erschaffen werden. Auch wenn diese im Hintergrund aus vielen autonomen Microfrontends besteht. Durch die einheitlichen Vorgaben und das stringente Design wird dem Benutzer ein stimmiges UI/UX Erlebnis ermöglicht.⁵³

Ein *Design System* kann durch einen *Style Guide* realisiert werden, welcher Styles in Form von Schriftarten, Farben und Grafiken vorgibt. Alternativ kann eine *Komponentenbibliothek* zur Verfügung gestellt werden, welche Elemente zum Einbinden in Microfrontends in Form von Formularelementen, Navigationselementen oder Anzeigeele-

⁵²Geers2020

⁵³Geers2020

menten liefert.⁵⁴ Ein *Design System* zu implementieren und zu Pflegen bedeutet jedoch Mehraufwand, welcher in Kauf genommen werden muss.

Zusammenfassend kann also gesagt werden, dass bei einer Microfrontend-Architektur die Vorteile Flexibilität, Unabhängigkeit und Übersicht nur durch Mehraufwand in der Organisation sowie Redundanz realisierbar sind. Die Nachteile sind durch Mehraufwand kompensierbar, was bei vorheriger Planung und Überprüfung der Architektur gut zu bewältigen ist.

Wirkungsbereiche von Microfrontend-Architekturen

Microfrontend-Architekturen besitzen Vor- und Nachteile. Diese besondere Form der Softwarearchitektur ist nicht für alle Arten von Softwarelösungen geeignet. Wo diese Architekturen empfehlenswert sind und wo sie nicht gerechtfertigt sind, wird nachfolgend erläutert.

Eine Microfrontend-Architektur bietet sich für große, verteilte Softwarelösungen an. Der Bedarf nach Flexibilität und Skalierbarkeit tritt erst ab einem gewissen Umfang der Softwarelösung auf. Bei kleineren Projekten sind Kosten und Schnelligkeit entscheidender.⁵⁵

Die angezeigten Frontends dynamisch auszutauschen funktioniert am besten clientseitig durch SPAs. Serverseitiges Rendering erfordert ein erneutes Laden des Inhaltes, was zwar technisch keine Herausforderung darstellt, jedoch nicht so flüssig und unbemerkt wie clientseitige Komposition abläuft. Smartphone Apps als Microfrontend zu realisieren ist schwieriger, da die Betreiber von App Stores strikte Kontrollen des Inhaltes durchführen, bei denen jedes Microfrontend in jeder Version überprüft werden müsste. Dadurch schwinden die flexiblen, unabhängigen Vorteile von Microfrontends.⁵⁶

Eine monolithische Anwendung in viele einzelne Teile aufzutrennen bringt Vorteile und Nachteile mit sich, welche im vorherigen Abschnitt erläutert wurden. Es muss vermehrter organisatorischer Aufwand in Kauf genommen werden. Dieser entsteht durch zusätzliche Abstimmung der Teams untereinander, bspw. für einheitliche Namespaces und Trennung der Verantwortung. Die Aufteilung einer monolithischen Anwendung in ein verteiltes System sorgt für zusätzliche Komplexität. Den verantwortlichen Teams

⁵⁴Geers2020

⁵⁵Geers2020

⁵⁶Geers2020

muss die gewonnene Flexibilität wichtiger sein, als der Aufwand und die Komplexität.⁵⁷

„Microfrontends are a sensible option when we are working on software that requires an iterative approach and long-term maintenance, when we have projects that require multiple teams to work on the same application, or when we want to replace a legacy project in an iterative way.“⁵⁸

Eine Microfrontend-Architektur ist nicht effektiv, wenn nur wenige Entwickler an der Softwarelösung arbeiten. Außerdem müssen die Domänen und Verantwortlichkeiten der Teams gut gegeneinander abgetrennt sein, weil es ansonsten zu Unsicherheiten und Verständnisproblemen der Teams untereinander kommen kann.⁵⁹

2.3.3 Microfrontend-Architektur aus organisatorischer Sicht

Eine Microfrontend-Architektur hat neben den technischen Aspekten auch organisatorische Unterschiede zu herkömmlichen Architekturen. Einige dieser Aspekte, wie vertikale Teams, notwendige Abstimmungen und geteiltes Wissen wurden bereits in den vorherigen Abschnitten beiläufig erwähnt und werden nachfolgend noch einmal strukturiert erklärt.

Organisation der Teams

Es existieren verschiedene Möglichkeiten die Teams bei einer Microfrontend-Architektur zu organisieren. In der nachfolgenden Abb. 8 wird zwischen ausschließlich im Frontend agierenden Teams, Full-Stack Teams und vollständig autonomen Teams unterschieden.

Beim monolithischen Ansatz ist ein großes Team für die ganze Anwendung verantwortlich. Bei *Frontend only* Teams sind es mehrere, kleine Teams, die jeweils einen bestimmten Bereich des Frontends betreuen. Dies hat den Vorteil, dass die Entwickler in den kleinen Teams sich besser mit den Fachlichkeiten auskennen.⁶⁰

Ebenfalls besteht die Möglichkeit die Teams vertikal aufzuteilen, wodurch jedes Team die Full-Stack Verantwortung für seinen Teilbereich hat. Dies hat den Vorteil, dass

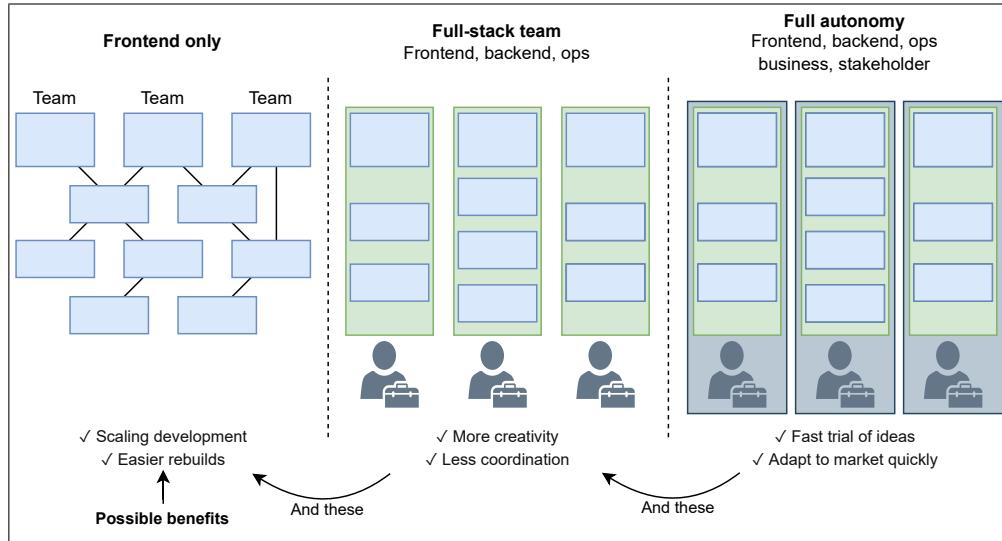
⁵⁷Geers2020

⁵⁸Mezzalira2021

⁵⁹Geers2020

⁶⁰Geers2020

Abbildung 8: Organisationsmöglichkeiten der Microfrontend Teams



Quelle: Geers2020

weniger Abstimmung notwendig ist, da Frontend und Backend der gleichen Domäne mitsamt der dazwischen liegenden Schnittstellen bei den selben Entwicklern liegen. Ebenfalls werden Full-Stack Teams aus Personen mit diversen fachlichen Wissensständen zusammengesetzt, was die Kreativität der Problemlösung und Qualität des Ergebnisses fördern kann.⁶¹

Werden die Stakeholder auch noch in das Full-Stack-Team integriert, so kann von einem vollständig autonomen Team gesprochen werden. Wenn die Stakeholder, wie bspw. Marketing, Kunden oder Support, direkt in den Problemlösungsprozess integriert werden, kann von noch kreativeren Blickwinkeln profitiert und die Time-to-Market (TTM) noch weiter reduziert werden.⁶²

Wissensverteilung

Das Wissen übergreifend über autonome Teams hinweg zu verteilen geschieht nicht automatisch, ist aber dennoch ein wichtiger Prozess. Die Mitglieder von vertikalen Teams treffen sich in sogenannten Community of Practice (COP), welche horizontal organisierten Expertenkreisen entsprechen. So können Best Practices und Erfahrungen der Teams in den notwendigen technischen Tiefen der jeweiligen Spezialisierungen ausgetauscht werden. In Anhang 2 in Abb. 51 sind drei autonome Teams dargestellt, welche sich

⁶¹Geers2020

⁶²Geers2020

regelmäßig in Expertenkreisen treffen und austauschen. So gibt es beispielsweise einen Expertenkreis für das Themengebiet Frontend, einen für die Datenanalyse und einen für Kubernetes als Infrastrukturplattform.

Das Konzept der COPs besteht schon länger, als die Microfrontend-Architektur. Doch es ist bei dieser Art der Teamorganisation besonders wichtig, da der sonst bestehende natürliche Austausch innerhalb eines horizontalen Teams verloren geht. In den COPs sollte sich auch auf Standards und Konventionen für Schnittstellen und Namespaces innerhalb eines Projektes geeignet werden.

Verwaltung des Codes

Bei klassisch monolithischen Anwendungen gibt es ein Repository für die ganze Applikation. Bei Microservices und -frontends kann es für jede Einheit ein eigenes Repository geben. Dies wird als *Polyrepo* bezeichnet. Alternativ kann aber auch für das ganze Projekt nur ein Repository genutzt werden, welches dann einem *Monorepo* entspricht.

Vorteile eines *Monorepos* sind, dass dadurch die Absprache der Teams untereinander vereinfacht wird, weil der Code des anderen Teams direkt zugänglich ist und eingebunden werden kann. Ebenfalls kann der Code einfacher einheitlich überarbeitet werden, da alle Dateien nebeneinander verfügbar sind.⁶³

Polyrepos haben den Vorteil, dass die Autonomie gestärkt wird. Jedes Microfrontend hat sein eigenes Repository und eigene CI/CD Pipelines angebunden. Ebenfalls können individuelle Branchingstrategien pro Team verwendet und versehentliche Abhängigkeiten vermieden werden, da Zugriffe zwischen Teams nur über definierte APIs möglich sind.⁶⁴

Das Konzept des *Monorepos* wird im Microfrontend-Umfeld kontrovers diskutiert. Einige Autoren sehen Vorteile durch Wiederverwendbarkeit und vereinfachte Abhängigkeiten.⁶⁵ Andere Autoren sehen es jedoch als Antipattern, da Code geteilt wird und somit Autonomie verloren geht.⁶⁶

⁶³ Mezzalira 2021

⁶⁴ Mezzalira 2021

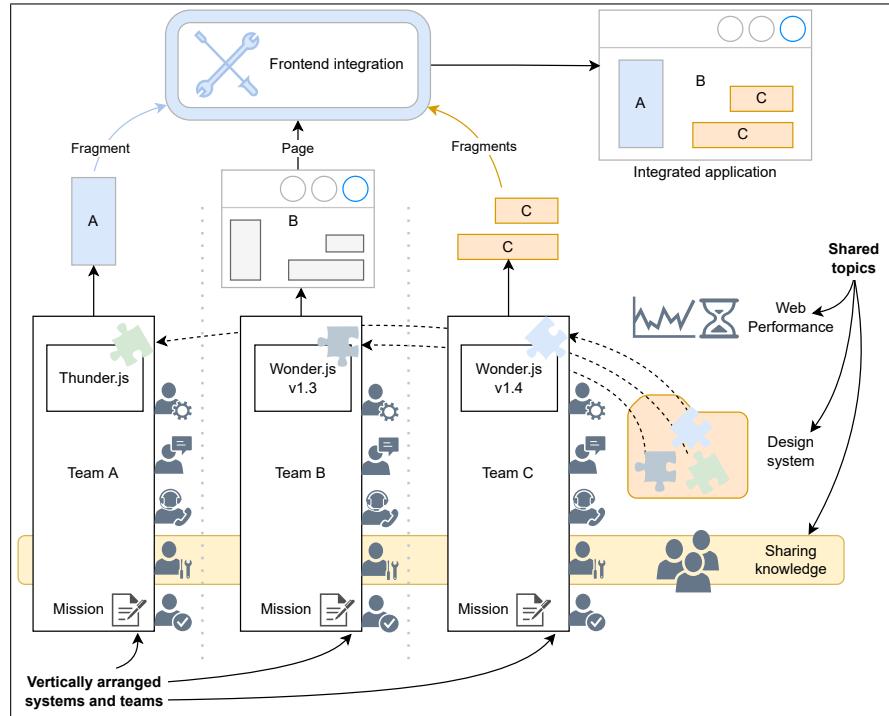
⁶⁵ Mezzalira 2021

⁶⁶ Geers 2020

Zusammenfassender Überblick

In der nachfolgenden Abb. 9 wird ein Gesamtüberblick über die Team-Organisation einer Microfrontend-Architektur gegeben.

Abbildung 9: Zusammenfassende Darstellung Microfrontends



Quelle: Geers2020

Es ist zu sehen, wie die einzelnen Microfrontends verschiedener, vertikal organisierter Teams zu einer zusammenhängenden Lösung zusammengefügt werden. Die vertikale Trennung nach Domänen, realisiert mit einem Team pro Domäne, ergänzt sich mit dem horizontalen Teilen von Wissen, Design- und Architekturentscheidungen. Auch können verschiedene Technologien je Team eingesetzt werden, was die Teams unabhängiger voneinander agieren lässt. Die Stakeholder der Domäne sind ebenfalls mit in den Entwicklungsprozess integriert, was das direkte Erstellen von Lösungen, passend zu den Anforderungen, ermöglicht.

2.4 Portalapplikationen

In diesem Kapitel wird das Konzept von Portalapplikationen erklärt. Portalapplikationen können in verschiedenen Formen realisiert werden. Im einfachsten Fall durch

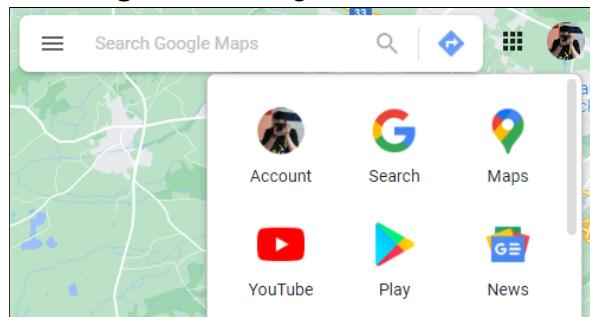
Hyperlinks, welche im nachfolgenden Abschnitt 2.4.1 vorgestellt werden. Ebenfalls ist die Realisierung einer Portalshell mit Querschnittsfunktionen möglich, welche im darunterliegenden Abschnitt erläutert wird.

Eine Portalapplikation stellt die übergeordnete Webseite dar, welche die einzelnen Microfrontends miteinander verbindet.⁶⁷ Dies kann auf unterschiedliche Arten realisiert werden, von denen zwei nachfolgend erläutert werden.

2.4.1 Portalapplikationen durch Hyperlinks

Portalapplikationen können bereits durch Verlinkungen realisiert werden. In der nachfolgenden Abb. 10 ist dies am Beispiel von Google Maps dargestellt. Die Seite <https://www.google.de/maps> bindet diverse andere Seiten des Alphabet Konzerns ein und macht diese durch Verlinkungen erreichbar.

Abbildung 10: Verlinkung anderer Webseiten



Quelle: Screenshot von Google Maps

Hinter jedem der im rechten Bereich der Abb. 10 zu sehenden Icons ist eine weitere Seite durch Klicken zu erreichen. So führt der Klick auf das YouTube Icon den Nutzer zu <https://www.youtube.com/>. Die bisherige Seite <https://www.google.de/maps> wird verlassen und der Seiteninhalt von YouTube wird dargestellt.

Der Vorteil bei dieser Art der Realisierung einer Portalapplikation besteht darin, dass Verlinkungen sehr einfach und ohne aufwändige Anpassungen einzurichten sind.

Allerdings verliert der Benutzer beim Wechseln der Webseiten seinen Fortschritt oder seine Eingaben. Ebenfalls muss Aufwand seitens der Webseitenbetreiber investiert wer-

⁶⁷Geers2020

den, damit die verlinkten Webseiten einheitlich aussehen, um die Zusammengehörigkeit sowie das Benutzergefühl zu erhalten.⁶⁸

Durch Hyperlinks kann die Produktpalette eines Unternehmens einheitlich erreichbar gemacht werden. Es können auch Informationen zu einem bestimmten Thema gesammelt werden, wie es beispielsweise bei <https://zeef.com/> der Fall ist. Eine Zeef-Seite zum Thema *Microfrontends* ist unter folgendem Link zu finden: micro-frontends.zeef.com.⁶⁹

2.4.2 Portalshell

Wenn das Wechseln zwischen verschiedenen Frontends nicht sichtbar geschehen soll, so muss eine Portalshell (engl. *Application Shell*) gewählt werden. Eine Portalshell ist eine Portalapplikation, welche andere Microfrontends nach horizontaler Trennung einbindet. In Anhang 2 in Abb. 58 ist eine beispielhafte Portalshell dargestellt, welche das clientseitige Routing zwischen verschiedenen Microfrontends übernimmt.

Die Portalshell hat mehrere Aufgaben, welche in der nachfolgenden Liste dargestellt sind:⁷⁰

- Dynamisch Microfrontend Routen laden
- *User state* verwalten
- Fehlerbehandlung beim Laden von Microfrontends
- Einheitliches Logging ermöglichen

Die Portalshell muss in der Lage sein, die Microfrontends dynamisch zu laden, damit beim Hinzufügen neuer Microfrontends keine neue Veröffentlichung der Portalshell erforderlich ist. Die Links zu den konfigurierten Microfrontends können hierfür bspw. in einer Datenbank abgelegt werden, auf welche die Portalshell zugreifen kann.⁷¹

Ebenfalls muss die Portalshell den *User state* kennen und bei Bedarf eine Authentifizierung durchführen. Die Portalshell muss verhindern, dass ein *Deep Link* aufgerufen werden kann, wenn keine Authentifizierung und Autorisierung vorliegen. Nachdem der

⁶⁸Steyer2020

⁶⁹Engel2021

⁷⁰Mezzalira2021

⁷¹Mezzalira2021

User authentifiziert wurde, muss die Portalshell das Aufrufen aller Microfrontends, für die er autorisiert ist, ermöglichen.⁷²

Sollte ein Microfrontend nicht geladen werden können, muss die Portalshell diesen Fehler abfangen und versuchen zu beheben. Unter Umständen ist das Microfrontend mehrfach auf verschiedenen URLs veröffentlicht und es kann auf eine andere Adresse gezeigt werden, die noch funktioniert. Das verantwortliche Team muss informiert werden und sollte das Problem schnellstmöglich beheben. Des Weiteren muss die Portalshell das Logging von Fehlern zentral ermöglichen. Das könnte theoretisch auch jedes Microfrontend selber realisieren, doch so kann Aufwand eingespart werden und wenn diese Informationen konsolidiert an einem Ort sind, ist die Fehlersuche für die betreuenden Teams einfacher.⁷³

Ebenfalls liefert die Portalshell den Microfrontends sogenannte Querschnittsaspekte. Diese können optional von den Microfrontends verwendet werden und bilden Funktionen ab, die sonst jedes Microfrontend selbstständig lösen müsste. So wird Redundanz vermieden und Einheitlichkeit gewahrt. Querschnittsaspekte sind nicht verbindlich und unterscheiden sich zwischen Portalshells, da sie von den individuellen Anforderungen abhängen.

Mögliche Querschnittsaspekte sind nachfolgend aufgelistet:⁷⁴

- User- und Mandantenverwaltung
- Authentifizierung & Autorisierung
- Lokalisierung
- Konfigurationswertespeicher

Die Portalshell verwaltet die Benutzer einer Anwendung und handhabt die Berechtigungen. Ein Benutzer wird anhand seiner Berechtigungen autorisiert. Eine Portalshell kann auch mandantenfähig sein. Das bedeutet, dass Benutzer zu einem oder mehreren Mandanten zugehörig sind. Spezielle Bereiche der Anwendung sind nur für einzelne Mandanten freigeschaltet und die Darstellung der Applikation ist für jeden Mandanten unterschiedlich. Beispielsweise entscheidet Mandant A, dass die zugehörigen Benutzer geduzt werden sollen, was die Ziel-Lokalisierung der Applikation beeinflusst.

⁷²Mezzalira2021

⁷³Mezzalira2021

⁷⁴vgl. Expertengespräch 1 in Anhang 4

Bei internationalen Portalapplikationen ist eine Lokalisierung empfehlenswert. So können mit einer Webseite direkt alle konfigurierten Sprachen abgebildet werden. Die Anzeigesprache ist pro User konfiguriert, sollte aber auch über das Frontend zu ändern sein, wie es beispielsweise bei einer großen Onlinehandelsplattform der Fall ist (siehe Abb. 46 in Anhang 2). Über die Lokalisierung kann zum einen die Sprache gewechselt werden, zum anderen aber auch der Dialekt einer Sprache. Letzteres empfiehlt sich bei einer deutschsprachigen, mandantenfähigen Portalshell, abhängig davon, ob der Mandant seine Benutzer siezt oder duzt.

Ebenfalls muss die Portalshell eine Möglichkeit für Microfrontends bieten, Konfigurationen abhängig vom User und dessen Mandanten im Key-Value Format laden zu können.⁷⁵ Als Beispiel wird der Name des aktuellen Mandanten immer in den Microfrontends angezeigt, was für jeden Mandanten dementsprechend unterschiedlich konfiguriert werden und dynamisch geladen werden muss.

Mandantenfähige Portalshells mit einer Nutzerverwaltung sowie einem Rechte- und Rollenkonzept sind anspruchsvoll zu gestalten. Ebenfalls muss der Prozess der Registrierung eines neuen Microfrontends, welches sich unter Umständen je Mandant unterscheidet, abgebildet werden. Im späteren Kapitel 3 wird eine beispielhafte Portalshell beschrieben, deren Prozesse im Detail erklärt werden.

2.5 Microfrontends

Im Abschnitt 2.3 wurde die übergreifende Architektur von Microfrontends erläutert. Im vorherigen Abschnitt 2.4 anschließend dann die Portalapplikationen, als gruppierendes Konzept von Microfrontends, erläutert. In diesem Abschnitt werden nun Microfrontends einzeln erläutert. Es werden verschiedenen Techniken vorgestellt, durch welche Microfrontends zusammengesetzt und eingebunden werden können.

In den beiden nachfolgenden Abschnitten 2.5.1 bis 2.5.2 werden die Konzepte von *Server-side* und *Client-side Composition* vorgestellt, welche bereits vorher in Abschnitt 2.3.2 beim Microfrontend Decision Framework angeschnitten wurden.

⁷⁵Mezzalira2021

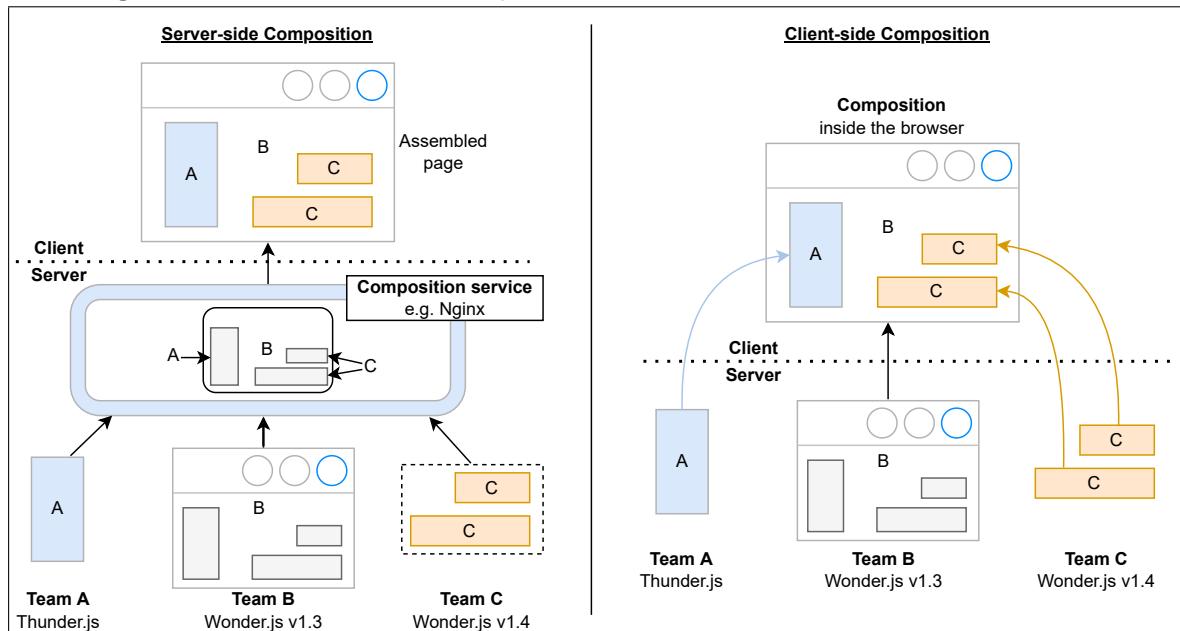
Danach werden in den Abschnitten 2.5.3 bis 2.5.5 verschiedene Arten der Einbindung eines Microfrontends in die Portalshell durch *Client-side Composition* erläutert.

Im letzten Abschnitt 2.5.6 wird eine Zusammenfassung der drei zusammengehörigen Abschnitte (*Iframe*, *Web Components* und *Module Federation*) bezogen auf das Thema Microfrontends gegeben. Die Auswirkungen der Entscheidungsvielfalt wird anschließend mit einer generischen Betrachtungsweise durch verschiedene Entscheidungsstrukturen aufgezeigt.

2.5.1 Server-side Composition

Bei der *Server-side Composition* werden die einzelnen Fragmente der Webseite auf dem Webserver zu einer fertigen Webseite zusammengesetzt und dann als ein vollständiges Stück an den Client herausgegeben. Dies ist in der nachfolgenden Abb. 11 im linken Teil dargestellt und geschieht auf dem Server durch die dort installierte Webserver-Lösung (bspw. Nginx).⁷⁶

Abbildung 11: Server- und Client-side Composition



Quelle: Geers2020

⁷⁶Geers2020

Dadurch, dass die gesamte Webseite auf dem Server zusammengesetzt und als vollständige Lösung an den Nutzer ausgegeben wird, sind gute Ladezeiten zu erreichen.⁷⁷

Bei Nginx, einer Softwarelösung für unter anderem Webserver, Cache, Lastverteilung und Reverse Proxy,⁷⁸ werden die einzelnen Fragmente durch *Server-Side Includes (SSI)* serverseitig zusammengesetzt. Die Seite, welche später an den Client ausgegeben wird, beinhaltet noch Platzhalter. Diese werden durch SSI mit dem Zieltinhalts ersetzt. In Anhang 1 unter Listing 11 ist ein exemplarischer Platzhalter dargestellt.

Die Platzhalter beinhalten den Pfad zum Code, durch welchen sie ersetzt werden sollen. Alle Platzhalter, die sich auf der auszugebenden Seite befinden, werden anschließend durch den referenzierten Code ersetzt. In der Abb. 41 in Anhang 2 ist ein Sequenzdiagramm dargestellt, welches die einzelnen Schritte im Detail darstellt.

Server-side Composition bietet Vorteile durch schnelle Performance, weil der Browser direkt eine fertig zusammengesetzte Webseite liefert bekommt. Ebenfalls ist SSI als Technologie bewährt und *Server-side Composition* ist suchmaschinenfreundlich.⁷⁹

Die Nachteile von *Server-side Composition* kommen vor allem bei großen, dynamischen Anwendungen zu Tage. Es dauert, bis alle Teile der großen Anwendung gerendert und zum Client übertragen werden. Eine Mischung von *Server-* und *Client-side Composition* könnte helfen, die Ladezeit zu verbessern. Ebenfalls wirken Server-side Applikationen nicht sehr interaktiv auf den Anwender. Programmteile die schnell reagieren und auf Nutzereingaben interagieren müssen, sollten für verringerte Ladezeiten clientseitig zusammengesetzt werden.⁸⁰

Server-side Composition ist für den weiteren Verlauf der Masterthesis nicht von Relevanz und wurde nur für die Gegenüberstellung zu *Client-side Composition* genutzt, welche im nachfolgenden Abschnitt 2.5.2 erklärt wird.

⁷⁷Geers2020

⁷⁸nginx2022

⁷⁹Geers2020

⁸⁰Geers2020

2.5.2 Client-side Composition

Bei der *Client-side Composition* werden die Fragmente einzeln heruntergeladen und erst beim Client im Browser zusammengesetzt. Auch dieser Vorgang ist in der vorherigen Abb. 11 dargestellt.

Im Gegensatz zu *Server-side Composition* werden bei der *Client-side Composition* die einzelnen Fragmente im Browser des Client generiert und zusammengesetzt. Lösungen dafür sind beispielsweise die gängigen Javascript-Frontend-Frameworks *Angular*, *Vue.js* oder *React*.

Je nach Art der Zusammensetzung können die Fragmente auch auf unterschiedlichen Technologien beruhen. So kann Fragment A auf einer neueren Version des Frameworks beruhen, welches die Fragmente zusammensetzt. Fragment B kann sogar auf einer ganz anderen Technologie beruhen, solange es die benötigten Referenzen und Bibliotheken eigenständig mitbringt.

Client-side Composition bietet sich bei interaktiven Anwendungen an, wie beispielsweise einer Web Applikation zum Erstellen der Steuererklärung. Eine statische Anwendung kann besser serverseitig zusammengesetzt und mit einzelnen clientseitigen Komponenten versehen werden, bei denen vorwiegend dynamischer Inhalt angezeigt wird.

Ebenfalls ist es durch Iframes und Web Components möglich, eine starke, technische Isolierung der UI zu erzielen.⁸¹ Dadurch können sich Fragmente verschiedener Teams in den Stylings nicht überschreiben, was bei *Server-side Composition* nur durch strikte Konventionen, wie beispielsweise Prefixe für CSS-Klassen und Namespaces, möglich ist.⁸²

In den nachfolgenden Abschnitten werden drei Arten der clientseitigen Einbindung eines Microfrontends vorgestellt.

2.5.3 Iframe

Das *Inline Frame* (nachfolgend Iframe) ist ein HTML Element, durch welches andere HTML-Webseiten in die bestehende Webseite eingebunden werden können. Es wird von

⁸¹Geers2020

⁸²Geers2020

allen aktuellen Browsern unterstützt.⁸³ Im nachfolgenden Listing 1 ist ein beispielhaftes Iframe zu sehen, über welches die Webseite der Fachhochschule der Wirtschaft (FHDW) eingebunden wird.

Listing 1: Exemplarische Einbindung einer Webseite über das Iframe-Element

```
<iframe id="iframeId" src="https://fhdw.de/" name="FHDW-Webseite"  
width="500" height="200" style="border: none"></iframe>
```

Quelle: Eigene Darstellung

Dem Iframe Element wird über den `src`-Parameter die URL der einzubindenden Webseite mitgegeben. Über die Parameter `width` und `height` können Anpassungen an der Breite und Höhe des Elements, in dem die eingebundene Webseite angezeigt wird, getätigt werden. Dies ist erforderlich, sollte die Standardauflösung von 300px x 150px nicht ausreichen.⁸⁴

Ebenfalls kann über den `style`-Parameter eigener CSS Code auf das HTML-Element angewendet werden. Durch diesen Parameter kann unter anderem der standardmäßige Rahmen ausgeblendet werden, um die Einbindung einer externen Ressource für den Anwender weniger offensichtlich zu machen.

Ein Iframe kann, wenn es in eine andere Anwendung eingebunden wird, diese nicht durch Styles beeinflussen. Die Kapselung der UI ist dementsprechend hoch gegenüber anderen Microfrontends.

Das Iframe eignet sich als schnelle Art der Einbindung mit hoher Kapselung des Inhalts gegen andere Microfrontends. Iframes haben eine schlechte Performance, wenn viele von ihnen eingebunden werden. Jedes Iframe erstellt einen neuen Browserkontext, welcher Rechen- und Speicherintensiv ist. Ebenfalls sind Iframes für Screenreader und Suchmaschinen nicht gut zu erreichen, was in geringer Suchmaschinenoptimierung und geringer Barrierefreiheit resultiert.⁸⁵

⁸³MDNWebDocs2021a

⁸⁴MDNWebDocs2021a

⁸⁵Geers2020

2.5.4 Web Components

Web Components bestehen aus drei Technologien: *Shadow DOM*, *Custom Elements* und *HTML Templates*.⁸⁶

Durch das *Shadow Document Object Model (DOM)* wird einem HTML-Element ein eigener, sogenannter Shadow, DOM Tree gehängt. Dieser Tree wird vom Browser separat und unabhängig vom Main DOM gerendert. Das separate Rendering der beiden DOMs hat zur Folge, dass diese voneinander abgekapselt agieren und sich nicht gegenseitig beeinflussen können. So haben CSS-Konfigurationen innerhalb des Shadow DOM Trees keine Auswirkungen auf Elemente außerhalb des eigenen Trees. Dadurch wird vermieden, dass eingebundener Code die einbindende Webseite beeinflusst und zugleich nicht von CSS-Konfigurationen der einbindenden Webseite beeinflusst wird.

Die Elemente im Shadow DOM zu Kapseln hat Vor- und Nachteile. Durch den eigenen DOM ist die Isolierung des Codes gegenüber anderen Microfrontends, wie bei Iframes auch, sehr stark. Ebenfalls können globale Styles die Web Component nicht beeinflussen, was bei legacy Anwendungen ein Vorteil ist. Auf der anderen Seite wird Shadow DOM in alten Browsern nicht unterstützt und die Kapselung der Styles ist mit einigen Design Frameworks nicht kompatibel. Beispiele für inkompatible Frameworks sind Twitter Bootstrap oder Angular Material.⁸⁷

Für wiederverwendbaren HTML Code werden *HTML Templates* verwendet. Ein Template wird über den HTML-Tag `template` definiert und der Inhalt dessen bleibt beim Laden der Seite erst einmal verborgen. Das Template kann später je nach Bedarf ein oder mehrere Male über JS nachgeladen werden. Ein beispielhaftes Template ist in Anhang 1 in Listing 12 dargestellt.

Durch *Custom Elements* wird ermöglicht, dass neben den vordefinierten HTML Tags noch weitere, individuelle HTML Tags der Webseite hinzugefügt werden können. Der Inhalt eines Templates kann zu einem *Custom Element* Tag hinzugewiesen werden. Die dahinterliegende *Custom Elements* API rendert aus dem individuellen HTML Tag dann anschließend den definierten Inhalt.⁸⁸

⁸⁶MDNWebDocs2022a

⁸⁷Geers2020

⁸⁸Rauber2021

Web Components sind ein anerkannter Webstandard, der direkt mit der Browser API interagieren kann. Die starke Isolation des Inhaltes bei Bedarf sorgt für eine robuste Einbindung.⁸⁹ Dafür benötigen Web Components auch JS clientseitig und unterstützen kein serverseitiges Rendering. Dies ist zwar über Umwege realisierbar, aber nicht der vorgesehene Standard, was bei Anforderungen an geringe initiale Ladezeiten von Nachteil ist.⁹⁰

2.5.5 Webpack Module Federation

Webpack ist ein OpenSource Modulbundler, welcher Module und deren Abhängigkeiten in statische Pakete umwandelt. Das Frontend Framework Angular benutzt Webpack zum Transpilieren, Kompilieren und Veröffentlichen von Quellcode.⁹¹

Mit der Major Version 5 von Webpack, welche im Oktober 2020 offiziell erschienen ist, wurde ein neues Feature namens *Module Federation* vorgestellt.⁹² Durch Module Federation sind zwei neue Plugins hinzugekommen: Das *ContainerPlugin* und das *ContainerReferencePlugin*.

Das *ContainerPlugin* ermöglicht asynchronen Zugriff auf Module, welche in einem Container zur Verfügung gestellt werden. Das *ContainerReferencePlugin* ermöglicht Referenzen zu anderen, entfernten Containern, wodurch Module aus diesen entfernten Containern importiert und genutzt werden können.⁹³

Die Entwickler von Webpack beschreiben zwei Anwendungsfälle. Mit dem Ersten können klassische Microfrontend Applikationen realisiert werden. So sind alle Unterseiten einer Webseite als eigene Container veröffentlicht, welche ihren Inhalt als Modul bereitstellen. Die zentrale Seite, beispielsweise die Startseite der Webseite mit Navigationseinträgen im Header- und Seitenbereich, referenziert als Application Shell durch das *ContainerReferencePlugin* alle Unterseiten und bindet deren Content ein. Des Weiteren stellt die Application Shell gängige Bibliotheken bereit, auf die die Unterseiten zugreifen können. Da die Bibliotheken geteilt werden ist sichergestellt, dass diese nur einmal geladen werden.⁹⁴

⁸⁹Geers2020

⁹⁰Geers2020

⁹¹Clow2018

⁹²Webpack2020a

⁹³Webpack2020b

⁹⁴Webpack2020b

Das unabhängige Einbinden der Seiten hat zur Folge, dass die Unterseiten alle einzeln neu veröffentlicht werden können, wenn neuer Inhalt für eine neue Version bereit steht. Die aufrufende Seite muss nur dann neu veröffentlicht werden, wenn Änderungen an den Links vorgenommen werden. Wird lediglich der Inhalt der Unterseiten geändert, ist keine neue Veröffentlichung der Hauptseite erforderlich.⁹⁵

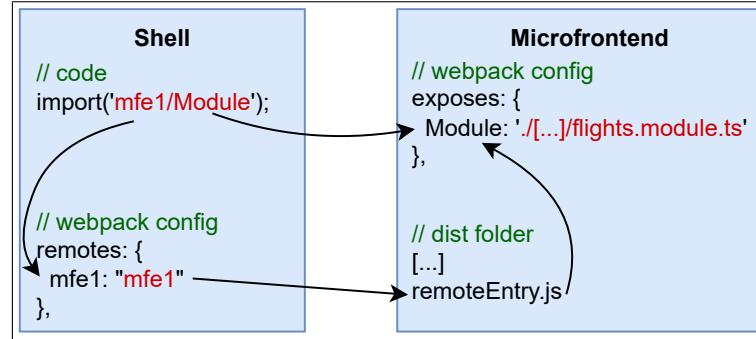
Beim zweiten Anwendungsfall wird eine Komponentenbibliothek, bspw. ein Software Development Kit (SDK), zum Einbinden zur Verfügung gestellt. Alle Apps, die mit diesem SDK arbeiten, integrieren es mittels Module Federation in ihre Applikation. Wenn eine neue Version des SDKs veröffentlicht werden soll, muss nur das Modul des *ContainerPlugins* neu veröffentlicht werden und nicht jede einbindende App.⁹⁶

Module Federation in Angular

Webpack ist der Modulbundler für Angular. Seit Angular Version 11 wird Webpack in der Major Version 5 unterstützt. Dadurch ist mit Module Federation ein weiterer Weg zur Einbindung eines Microfrontends in eine Portalshell hinzugekommen. Module Federation ermöglicht es, Programmteile zu referenzieren die zur Compilezeit noch nicht bekannt sind. Ebenfalls können sich verschiedene Microfrontends Bibliotheken teilen, sodass diese nur einmal vom Client geladen werden müssen.⁹⁷

Damit eine Applikation mittels Module Federation eingebunden werden kann, müssen auf beiden Seiten Konfigurationen vorgenommen werden. In der nachfolgenden Abb. 12 sind die notwendigen vier Schritte zur Einbindung dargestellt.

Abbildung 12: Notwendige Konfigurationen für Module Federation



Quelle: Steyer2020a

⁹⁵Webpack2020b

⁹⁶Webpack2020b

⁹⁷Steyer2020

Jede Applikation, die Module Federation einsetzt, muss über eine `webpack.config.js`-Datei verfügen. Über diese Datei wird Webpack konfiguriert und die Module der Applikation aufgelistet, welche exportiert und importiert werden sollen. Die Portalshell referenziert ein entferntes Modul. Dieses Modul wird von einem Microfrontend exportiert und in die `remoteEntry.js`-Datei im *dist*-Ordner von Webpack geladen. Der Pfad zu diesem Ordner ist in der Konfiguration der Shell hinterlegt.

In Listing 3 in Anhang 1 ist der relevante Teil einer `webpack.config.js`-Datei für eine RemoteApp zu sehen. Die RemoteApp veröffentlicht in der `remoteEntry.js`-Datei das Angular Distant-Modul mit dem Namen *microapp*.

In Listing 2 in Anhang 1 ist der dazugehörige relevante Teil einer `webpack.config.js`-Datei einer Shell zu sehen. Die Shell bindet das RemoteModul *microapp* ein, welches unter `microapp@http://localhost:3000/remoteEntry.js` zu finden ist. Das @-Zeichen trennt dabei den Namen des Microfrontends von der URL des Microfrontends.

Ebenfalls sind in der Konfiguration der Shell die Bibliotheken zu sehen, welche für die RemoteApps zur Verfügung gestellt werden. Im Beispiel von Listing 2 in Anhang 1 werden die vier gängigen Angular Bibliotheken `@angular/core`, `@angular/common`, `@angular/common/http` und `@angular/router` zur Verfügung gestellt. Das Microfrontend hat die gleichen Bibliotheken als geteilt konfiguriert und greift auf die Bibliotheken der Shell zurück, anstatt sie selbst mitzubringen.

Damit die Shell mit dem Microfrontend Daten austauschen kann, muss in den Konfigurationsdateien beider Seiten ein `SharedMapping`-Objekt referenziert werden. Dort kann eine individuelle Angular Library⁹⁸ eingebunden werden, welche von der Shell befüllt und vom Microfrontend konsumiert wird.⁹⁹

Module Federation mit Web Components

Mit Module Federation ist es ebenfalls möglich, neben Angular Modulen und Komponenten, auch Web Components zu Teilen. Dies ist dann sinnvoll, wenn die einbindende Shell über eine andere Framework Version als das Microfrontend verfügt.

Dafür muss das Microfrontend in der `app.module.ts` die zu exportierende Angular Komponente als `customElement` definieren, analog wie in Abschnitt 2.5.4 beschrieben.

⁹⁸Angular2022h

⁹⁹Steyer2021d

In der Webpack Konfiguration muss anstelle des Modules die `bootstrap.ts` veröffentlicht werden. Statt eines Modules wie bei der vorherigen Variante, lädt die Portalshell ein Script von dem RemoteEntry.¹⁰⁰

Durch das Einbinden einer Web Component können Microfrontends einer anderen Angular Major Version geladen und geteilt werden. Ebenfalls können Web Components eingebunden werden, die in einem anderen Web Frontend Framework erstellt wurden (bspw. in React). In Abb. 52 in Anhang 2 sind Schnittmengen von vier Microfrontends mit einer Shell skizziert. Bibliotheken mit gleicher Major Versionsnummern können geteilt werden, wie es bei Microfrontend 2 (MFE2) und Microfrontend 3 (MFE3) der Fall ist. Module Federation wählt automatisch die Bibliothek mit der höheren Minor Version und teilt sie zwischen Shell und Microfrontend. Microfrontends mit anderen Web Frameworks, wie das React Microfrontend 4 (MFE4) in der Abbildung, müssen vollständig zum Client übertragen werden.

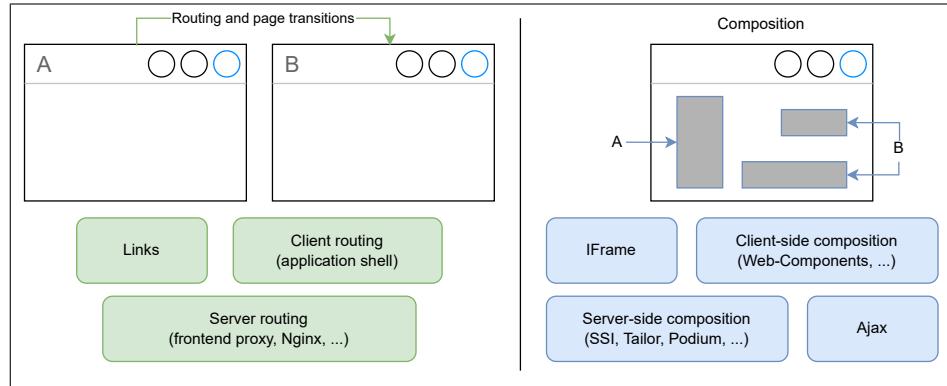
2.5.6 Auswirkungen der Entscheidungsvielfalt

In diesem abschließenden Abschnitt sollen die Auswirkungen der großen Entscheidungsvielfalt beim Thema Microfrontends verdeutlicht werden. In den vorherigen Abschnitten wurde auf die Architektur von Microfrontends, Portalapplikationen und verschiedene Microfrontend-Varianten eingegangen. Je nachdem, welche Entscheidungen Softwarearchitekten für die Applikationen treffen, ergeben sich unterschiedliche Konsequenzen für die verwendeten Technologien.

Die Softwarearchitekten, welche eine Microfrontend-Architektur auswählen, müssen verschiedene Entscheidungen treffen. Diverse Lösungsarten für Komposition und Routing sind in der nachfolgenden Abb. 13 dargestellt.

¹⁰⁰Steyer2021a

Abbildung 13: Weiterleitung und Zusammensetzung von Microfrontends



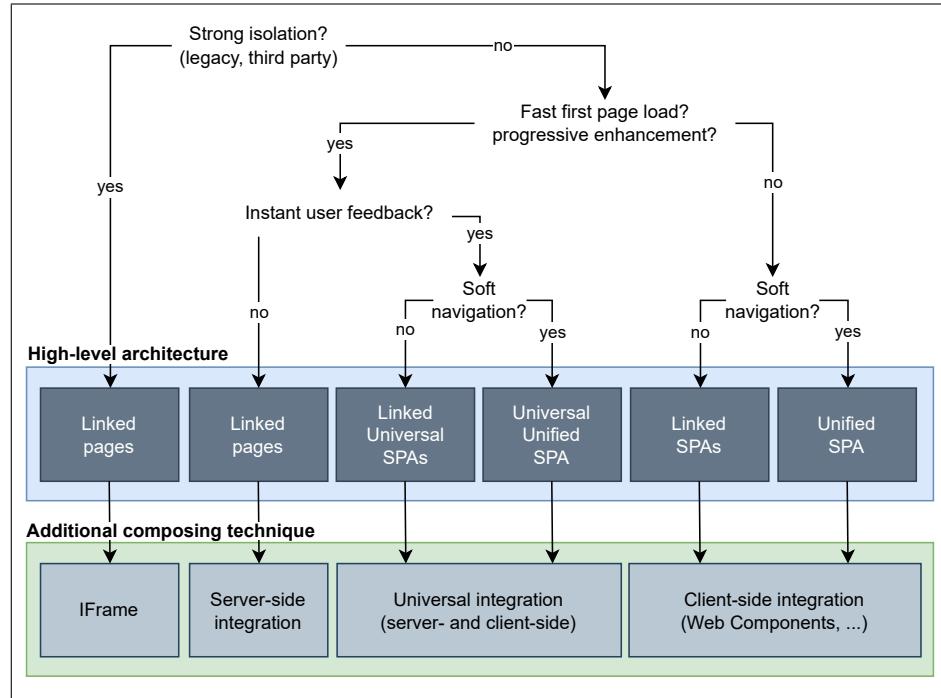
Quelle: Geers2020

Wenn durch das Routing die Seite wechselt oder sichtbar neu geladen werden muss, wird nachfolgend von *hartem Routing* gesprochen. Das Gegenstück dazu ist *softes Routing*, bei welchem die Seiteninhalte dynamisch ohne Übergang ausgetauscht werden. Das Routing bestimmt das Benutzererlebnis und kann mit hartem Routing durch Links realisiert werden, oder aber auch client- oder serverseitig geschehen. Bei den gängigen Web Frontend Frameworks sind Lösungen für softes Routing enthalten, wie beispielsweise der *Angular Router* bei Angular. Softes Routing ist für den Benutzerkomfort zu empfehlen, wohingegen ein hartes Routing durch Links den Wechsel in eine andere Webseite der Produktpalette eines Unternehmens signalisiert.

Das Einbinden der Microfrontends in eine Shell kann durch verschiedene Techniken realisiert werden. Alle haben Vor- und Nachteile. Es muss anhand des gegebenen Anwendungsfalles jeweils individuell entschieden werden, welche dieser Techniken zum Einsatz kommen.

Ein möglicher Entscheidungsbaum ist nachfolgend in der Abb. 14 dargestellt. Die Abbildung bietet anhand der Anforderungen eine Entscheidungshilfe bei der Auswahl der Kompositionstechnik des Frontends.

Abbildung 14: Microfrontend-Architektur Entscheidungsbaum nach Geers



Quelle: Geers2020

Die Entscheidung zur Art der Einbindung hängt davon ab, welche Isolation gewünscht ist, wie sich die App beim Navigieren verhalten soll und ob Wert auf initiale Ladezeiten gelegt wird.

Andere Autoren haben ähnliche Entscheidungsbäume veröffentlicht. So hat beispielsweise Manfred Steyer 2018 einen Leitfaden zu Microfrontends anhängig von Architekturzielen veröffentlicht (siehe Abb. 42 in Anhang 2), welcher allerdings Module Federation noch nicht beinhaltet.¹⁰¹

¹⁰¹ Steyer2018

3 Prototypisches Beispiel in Angular

In diesem Kapitel wird ein prototypisches Beispiel beschrieben, auf welches sich die im nachfolgenden Kapitel 4 stattfindende Evaluierung beziehen wird. Das Ergebnis der Evaluierung wird im darauffolgenden Kapitel 5 angewendet, um die Ergebnisse zu verifizieren.

Zum Verständnis des Prototypen wird in den nachfolgenden Abschnitten zunächst die Architektur in einer Schichtenansicht präsentiert und anschließend detaillierter auf die Portalshell sowie die eingebundenen Microfrontends eingegangen. Zuletzt werden in Abschnitt 3.4 die Anforderungen der Microfrontends für die Einbindung in die Portalshell konkret benannt, damit diese in der Evaluierung im darauffolgendem Kapitel berücksichtigt werden können.

Szenario des Beispieles

Bei der nachfolgend beschriebenen Beispielapplikation handelt es sich um eine Portalshell, welche im B2B-Kontext verwendet wird. Firmenkunden wird Zugang zu der Portalshell gewährt und diese können über die Softwarelösung ihre Backoffice Tätigkeiten erledigen. Die Portalshell ist mandantenfähig, denn jeder B2B Kunde soll individuell konfigurierte Microfrontends nutzen können.

Damit die Portalshell nicht bei jedem hinzugefügten Microfrontend jedes Mandanten neu veröffentlicht werden muss, ist das Anzeigen der Microfrontends dynamisch gelöst. Es werden von den Usern die individuellen Konfigurationen und Anordnungen der Microfrontends zur Laufzeit geladen. Als Beispiel möchte Kunde A auf der Startseite ein Microfrontend zum Ticketmanagement angezeigt bekommen. Kunde B hingegen benötigt auf der Startseite zwei konfigurierte Key Performance Indicator (KPI)s sowie ein Microfrontend zur Suche auf dem eigenen Fileshare.

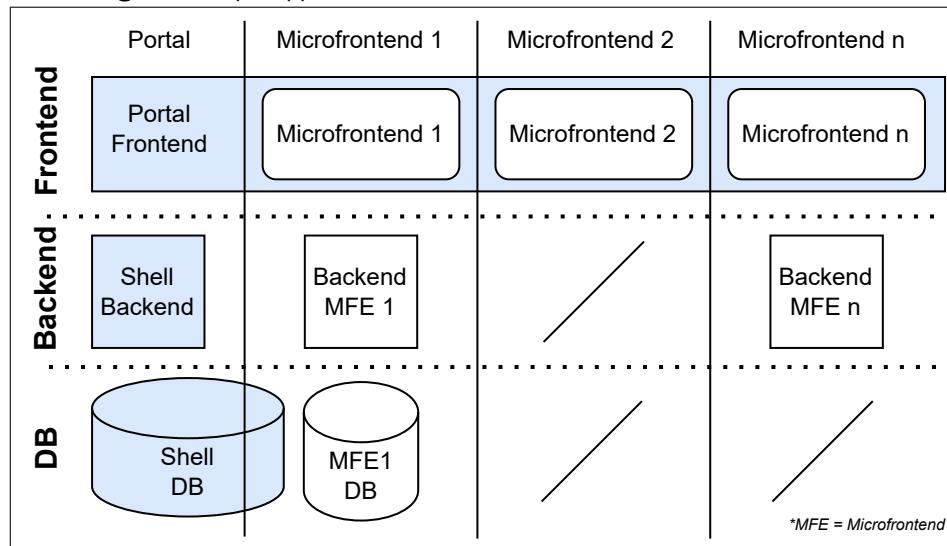
Die Portalshell bietet den Microfrontends, welche auch teilweise vom B2B Kunden selber entwickelt werden, die Möglichkeit auf Querschnittsasspekte des Portals zuzugreifen. Darunter fallen beispielsweise Logging, Lokalisierung, Benutzer- und Mandanteninformationen, Autorisierung und das Laden von Konfigurationswerten. Die Portalshell wird von mandantenumabhängigen Administratoren übergreifend verwaltet. Sie können neue Microfrontends registrieren, User anlegen und diese zu Mandanten zuweisen sowie einen First-Level-Support stellen. Die Administratoren sind ebenfalls für den Betrieb der Portalshell zuständig und Hosten diese in Azure.

In den nachfolgenden Abschnitten wird die Architektur sowie Details zu drei prototypischen Microfrontends für einen Mandanten im Detail präsentiert.

3.1 Architekturübersicht

In der nachfolgenden Abbildung Abb. 15 ist die beispielhafte Microfrontendarchitektur dargestellt und in die Schichten Frontend, Backend und Datenbank getrennt.

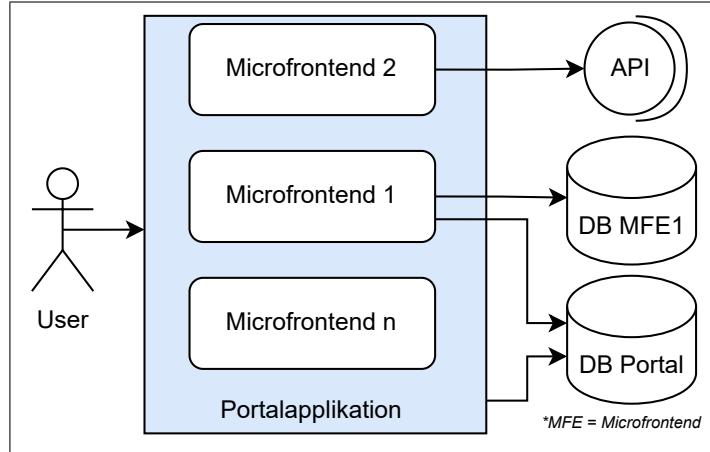
Abbildung 15: Beispielapplikation Microfrontend-Architektur



Quelle: Eigene Darstellung

Die Applikation besteht aus einer Portalshell, zu welcher ein Frontend, ein Backend sowie eine Datenbank gehört. In das Frontend der Portalshell sind mehrere Microfrontends eingebunden, welche entweder aus einem Frontend mit einem eigenen Backend sowie einer eigenen Datenbank bestehen (siehe Microfrontend 1 (MFE1)), nur aus einem Frontend bestehen (siehe MFE2) oder aus einem Frontend und einem Backend bestehen (siehe Microfrontend n).

Das Backend der Portalshell sowie das Backend des MFE1 greifen jeweils auf die Datenbank der Portalshell zu (siehe nachfolgende Abb. 16). Die Portalshell verwaltet die Nutzer und deren Berechtigungen. Sie stellt Schnittstellen bereit, damit Microfrontends die Nutzerinformationen bei Bedarf konsumieren können.

Abbildung 16: Beispielapplikation Solutionarchitektur

Quelle: Eigene Darstellung

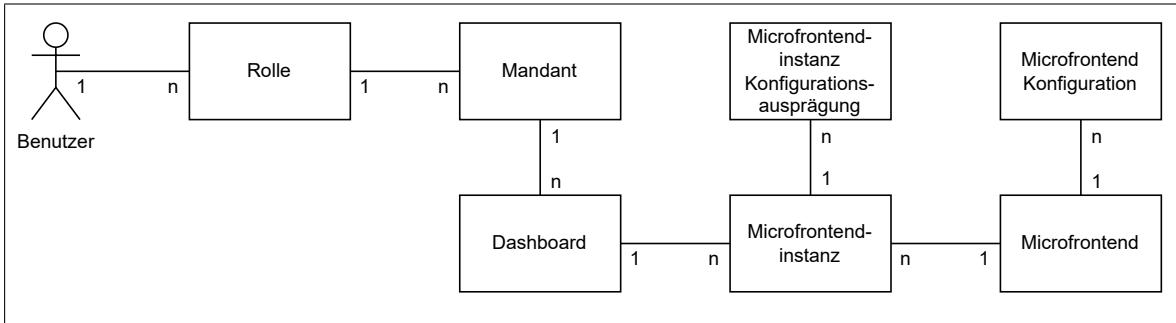
Die Portalapplikation verfügt über ein Rollenkonzept. Die Rollen werden für jeden User pro Microfrontend vergeben. Das Microfrontend 1 fragt die Datenbank (DB) der Portalshell nach der Rolle des aktuellen Users ab, weil es diese Information zum Anzeigen der Inhalte benötigt. Bei Microfrontend n und MFE2 haben alle Benutzer die gleichen Rechte, weswegen Abfragen der Rollen nicht nötig sind. Das MFE2 fragt in regelmäßigen Abständen eine öffentliche API an und zeigt die Ergebnisse der Anfrage aufbereitet im Frontend an.

3.2 Portalapplikation

Die Portalapplikation basiert auf dem Angular Framework und bietet, wie in Abschnitt 2.4 beschrieben, einige Querschnittsaspekte.

So ist Authentifizierung sowie die generelle Benutzerverwaltung über das Portal abgedeckt. Für Benutzer muss, bevor sie sich im Portal anmelden können, ein Account zum Einloggen sowie eine dazugehörige Rollenzuweisung angelegt werden. Es sind zwei Rollen verfügbar, welche in der Tabelle 8 in Anhang 3 mit ihrer jeweiligen Funktion dargestellt sind.

Eine Rollenzuweisung stellt die Verbindung zwischen einem Benutzer und einem Mandanten her. Ein Benutzer kann in mehreren Mandanten jeweils eine Rolle zugewiesen bekommen. Eine Übersicht der zusammenhängenden Elemente in der Portalapplikation ist in der nachfolgenden Abb. 17 zu sehen.

Abbildung 17: Übersicht der Elemente der Portalshell

Quelle: Eigene Darstellung

In dem Portal können Microfrontends dynamisch eingebunden werden. Es gibt eine Oberfläche, auf der Administratoren des Portals neue Microfrontends konfigurieren können. Zu jedem Microfrontend wird ebenfalls die Konfiguration der Parameter des Microfrontends gespeichert. Microfrontends können anschließend instanziert werden, indem sie einem Mandanten zugewiesen und die Parameter-Konfigurationen auf die Werte des Mandanten gesetzt werden. So können die Microfrontends dynamisch für verschiedene Mandanten gleichzeitig genutzt werden. Eine Übersicht der Elemente mit beispielhaften Ausprägungen ist zur besseren Verständlichkeit in Anhang 2 in Abb. 56 dargestellt.

Ein Dashboard bezeichnet eine für einen Mandanten individuell konfigurierte Anordnung von Microfrontendinstanzen, auf die Benutzer eines Mandanten zugreifen können. Die einzelnen Microfrontendinstanzen können für den eingebundenen Mandanten individuell konfiguriert sein. Liegt keine Konfiguration vor, wird entweder auf einen Standardwert des Microfrontend-Templates zurückgegriffen oder ganz auf Konfiguration verzichtet.

Die Nutzer eines Mandanten können sich die eingebundenen Microfrontendinstanzen individuell auf dem Dashboard konfigurieren, wie es zum Beispiel auch bei Azure Dashboards der Fall ist.¹⁰² Die Portalshell stellt Dialoge zum Registrieren und Verwalten von Dashboards für einen Mandanten zur Verfügung.

Damit die Portalapplikation sowie die eingebundenen Microfrontends integriert werden können und dem gleichen Design folgen, wird ein SDK zur Verfügung gestellt. Dieses beinhaltet Formularelemente, wie beispielsweise Buttons, interaktive Textboxen oder

¹⁰²Microsoft 2021a

Dropdownlisten. Diese Formularelemente sind einheitlich gestylt und können miteinander interagieren, um clientseitige Validierung zu ermöglichen.

Ebenfalls sorgt die Portalshell für Fehlerbehandlung, sollte ein Microfrontend in einem Dashboard nicht geladen werden können. In diesem Fall wird ein Platzhalter mit entsprechendem Text angezeigt und der Fehler protokolliert, damit das Entwicklerteam reagieren kann. Auch bietet das Portal eine Schnittstelle, damit eingebundene Microfrontends etwaige Fehler zentral protokollieren können.

3.3 Eingebundene Microfrontends

In die im vorherigen Abschnitt beschriebene Portalapplikation werden Microfrontends eingebunden. Um ein Microfrontend einbinden zu können, muss dieses vorher im Portal mit den Parametern URL, Art der Einbindung und optionalen mandantenindividuellen Konfigurationen registriert werden. Es entsteht ein Template des Microfrontends, welches nur die Hosting-Parameter sowie die Schlüssel der Konfigurationen enthält. Die Ausprägungen der Konfigurationen werden bei der Instanziierung des Microfrontends zu einem Mandanten gesetzt. Anschließend lädt die Portalshell die Konfiguration der Microfrontendinstanz dynamisch aus der Datenbank, damit sie nicht immer bei jeder Änderung an den konfigurierten Microfrontends neu veröffentlicht werden muss.

Im Fall der in Abb. 15 beschriebenen Architektur handelt es sich bei MFE2 um eine Wetter-Applikation, welche regelmäßig eine Wetter-API aufruft. Für jeden Mandanten ist der Standort als Parameter der Microfrontendinstanz konfiguriert, sodass das Wetter des Standortes angezeigt wird, an welchem der Mandant sich befindet. Exemplarisch soll für den prototypischen Mandanten das Wetter von zwei Standorten angezeigt werden, da Benutzer das Wetter an ihrem Arbeitsplatz, aber auch von ihrer Heimatstadt verfolgen können sollen. Von dem Wetter-Microfrontend werden dementsprechend zwei Instanzen mit unterschiedlichen Parametern eingebunden.

Microfrontend n ist eine Taschenrechner App, welche von allen Usern genutzt werden darf. Die Taschenrechner-App verfügt über keine Parameter und ist für alle Mandanten gleich.

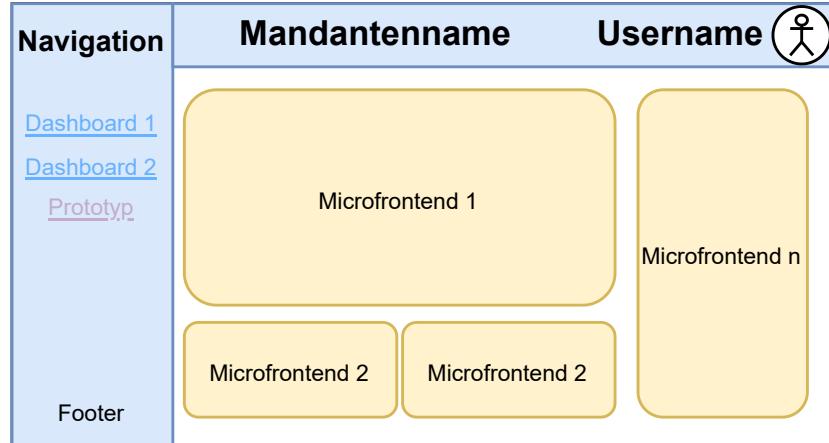
Microfrontend 1 stellt eine Statistik App dar. Diese darf von allen Benutzern angezeigt werden. Normale Benutzer sehen die App mit verringertem Umfang. Nutzer mit der

Administrator-Rolle sehen erweiterte Statistiken, die sonst nicht sichtbar sind. Die Statistik-App fragt die Portalapplikation an, ob der Benutzer ausreichend Rechte hat, um den Administrator-Bereich der Statistik-App einsehen zu dürfen. Dafür muss die Statistik-App Informationen über den aktuellen Benutzer kennen, wie beispielsweise seinen Namen und eine eindeutige Kennung.

Die externen Abhängigkeiten der Microfrontends sind in einer Solutionarchitektur dargestellt, welche in Anhang 2 in Abb. 28 abgebildet ist.

Ein Mockup des Designs von der Portalshell mit den vier eingebundenen Microfrontendinstanzen auf einem Dashboard mit dem Namen *Prototyp* ist in der nachfolgenden Abb. 18 dargestellt. Die blauen Balken links und oben werden von der Portalshell gestellt. Links an der Seite befindet sich die Navigation, durch die zwischen Dashboards gewechselt werden kann. Oben im Header sind Informationen zum Mandanten sowie ein Icon sichtbar, auf welches der Benutzer zum Verwalten seines Profils und zum Abmelden klicken kann.

Abbildung 18: Design Mockup Dashboard *Prototyp*



Quelle: Eigene Darstellung

Der restliche in weiß dargestellte Bereich der rechteckigen Anwendung ist der Platzhalter des ausgewählten Dashboards. Die vier eingeblendeten Microfrontendinstanzen entstammen drei konfigurierten Microfrontend-Templates.

3.4 Anforderungen an das System

Nach dem Definieren der Portalapplikation sowie den eingebundenen Microfrontends können konkrete Anforderungen gestellt werden, welche die Art der Einbindung erfüllen sollte.

Durch das in Abb. 17 aufgezeigte und in Abschnitt 3.3 beschriebene Konzept der Konfigurierbarkeit der Microfrontends muss bei der Art der Einbindung eine Möglichkeit bestehen, Parameter übergeben zu können. Ansonsten wäre eine mandantenfähige Portalapplikation nicht zu realisieren.

Da die UX das Erlebnis des Benutzers auf der Webseite beschreibt, spielt sie eine relevante Rolle. So sollten Portalapplikation und Microfrontends ein einheitliches Design haben und insgesamt über geringe Ladezeiten verfügen.

Weiterhin soll der Benutzer nach Möglichkeit im Portal bleiben und nicht zwischen verschiedenen Webseiten wechseln müssen. Es soll sich anfühlen, als wäre er durchgehend nur in der Portalapplikation unterwegs. Softes Routing sollte der Standardfall sein.

In sinnvollen Ausnahmefällen, beispielsweise bei einem Klick auf *Mail senden*, dürfte auch ein hartes Routing in ein anderes Programm erfolgen. Ebenfalls soll es möglich sein mehrere Microfrontends auf einer Seite einzubinden. Der Nutzer muss in der Lage sein, sich die Dashboards individuell anzupassen und dort unter Umständen auch viele Instanzen des gleichen Microfrontends mit unterschiedlichen Parametern nebeneinander darzustellen.

Zusammengefasst sind für die exemplarische Angular Portalapplikation und die dazugehörigen eingebundenen Microfrontends die Aspekte Parameterübergabe, einheitliches Design, eingebundene Navigation und geringe Ladezeiten wichtig.

4 Evaluierung verschiedener Arten der Einbindung von Microfrontends

In diesem Kapitel werden Kriterien festgelegt, anhand derer Arten der Einbindung von Microfrontends in die Portalshell bewertet und verglichen werden können. Dies geschieht durch eine Nutzwertanalyse, deren theoretische Grundlagen zunächst im nachfolgenden Abschnitt 4.1 erläutert werden.

Anschließend werden in den Abschnitten 4.2 bis 4.5 die vier relevanten Schritte einer Nutzwertanalyse am Thema durchlaufen: Identifizieren der Kriterien, Gewichten der Kriterien, Nutzwerte je Alternative bestimmen und Nutzwerte untereinander vergleichen.

Im letzten Abschnitt 4.6 werden die Ergebnisse der Nutzwertanalyse interpretiert und auf Anwendungsszenarien übertragen.

4.1 Theoretische Grundlagen einer Nutzwertanalyse

In diesem Abschnitt werden die theoretischen Grundlagen einer Nutzwertanalyse erläutert, welche in den nachfolgenden Abschnitten des Kapitel 4 angewendet werden, damit ein Entscheidungsproblem zwischen mehreren Auswahlalternativen fundiert gelöst werden kann.

Nach Christof Zangemeister ist die Definition einer Nutzwertanalyse „Die Analyse einer Menge komplexer Handlungsalternativen mit dem Zweck, die Elemente dieser Menge entsprechend der Präferenzen des Entscheidungsträgers bezüglich eines multidimensionalen Zielsystems zu ordnen. Die Abbildung dieser Ordnung erfolgt durch die Angabe der Nutzwerte (Gesamtwerte) der Alternativen.“¹⁰³

In der Theorie nach Arnim Bechmann besteht die Nutzwertanalyse im Detail aus zehn Arbeitsschritten, welche nachfolgend in Tabelle 2 dargestellt sind.

¹⁰³Zangemeister1971

Tabelle 2: Arbeitsschritte einer Nutzwertanalyse

Nr	Schritt
1	Problemformulierung
2	Aufstellung des Zielsystems
3	Angabe der zu bewertenden Alternativen A_1, \dots, A_m
4	Bestimmung der Bewertungskriterien K_1, \dots, K_n aufgrund des Zielsystems und der (Objekt-) Alternativen
5	Messung der Zielerträge $k_{11}, \dots, k_{ij}, \dots, k_{nm}$
6	Skalierung der Zielerträge in die Zielerfüllungsgrade $e_{11}, \dots, e_{ij}, \dots, e_{nm}$
7	Festlegung der Kriteriengewichte g_1, \dots, g_n
8	Berechnung der Teilnutzen N_{ij} nach der Formel dargestellt in Gleichung (1)
9	Addition der Teilnutzen einer Alternative zum Nutzwert N_j dieser Alternative
10	Angabe der Rangordnung der Alternativen aufgrund der Nutzwerte

Quelle: Bechmann1978

In der nachfolgenden Gleichung (1) ist die Formel zur Berechnung der Teilnutzwerte aus dem achten Schritt der vorherigen Tabelle zu sehen.

$$N_{ij} = g_i \cdot e_{ij} \quad (1)$$

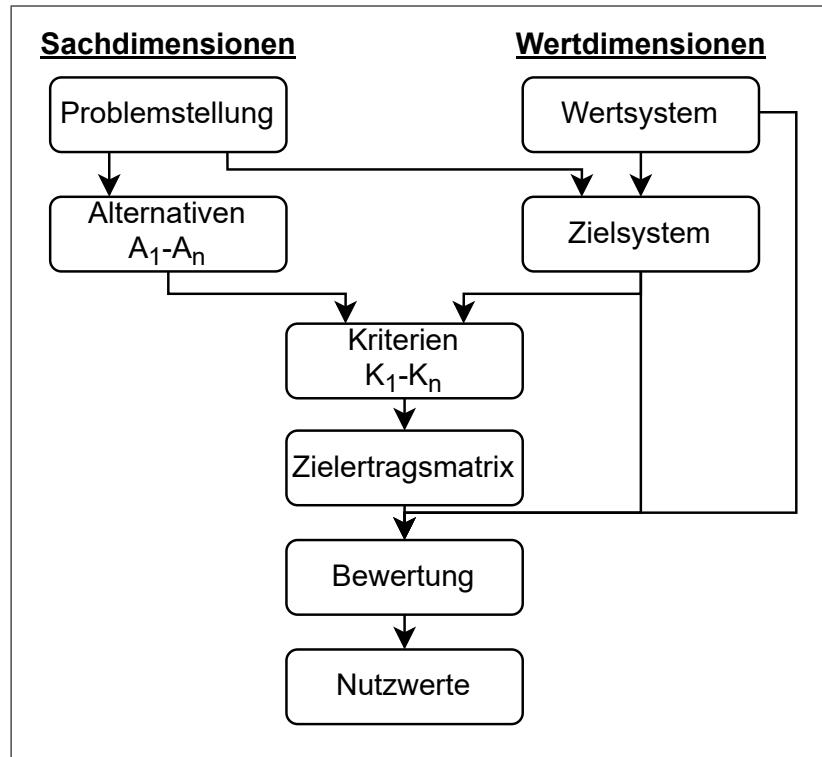
N_{ij} = Teilnutzen des i-ten Kriteriums der j-ten Alternative

g_i = Gewichtung des i-ten Kriteriums

e_{ij} = Zielerfüllungsgrad des i-ten Kriteriums der j-ten Alternative

Die zehn Schritte aus Tabelle 2 sind in einem Ablaufschema in der nachfolgenden Abb. 19 dargestellt. Das Ablaufschema stellt die Zusammenhänge der Einflussfaktoren zum Ergebnis, in diesem Fall die Nutzwerte, dar.

Abbildung 19: Allgemeines Ablaufschema einer Nutzwertanalyse



Quelle: Bechmann1978

Neben der Problemstellung und der für eine Nutzwertanalyse erforderlichen Lösungsalternativen fließen ebenfalls Entscheidungsregeln und Vorlieben des Autors als Wertesystem, sowie die Vorstellungen vom Zielsystem mit in die Bewertung ein.¹⁰⁴

„Die zu bewertenden Alternativen werden bezüglich dieses Zielsystems durch Bewertungskriterien beschrieben.“¹⁰⁵ Die Kriterien werden untereinander anhand ihrer Bedeutung gewichtet, damit wichtigere Kriterien mit höherem Anteil in den Nutzwert eingehen.

Die Gewichtung der Kriterien erfolgt durch die Paarvergleichsmethode. Jeder Teilnehmer der Nutzwertanalyse vergleicht jedes Kriterium gegeneinander. Es muss beim direkten Vergleich immer eine Entscheidung getroffen werden, welches Kriterium der beiden Verglichenen wichtiger ist. Zwei Kriterien dürfen nicht als gleich wichtig gesehen werden.¹⁰⁶

¹⁰⁴Bechmann1978

¹⁰⁵Bechmann1978

¹⁰⁶KÄijhnapfel2021b

Jeder Teilnehmer hat anschließend die Hälfte der nach der Gaußschen Summenformel (siehe nachfolgende Gleichung (2)) errechneten Vergleiche durchgeführt, wobei N für die Anzahl der Kriterien steht. Es wird nur die Hälfte durchgeführt, weil sich die andere Hälfte der Vergleiche automatisch bedingt. Denn ist Kriterium 1 (K1) besser als Kriterium 2 (K2), so ist K2 zwangsläufig schlechter als K1.

$$\sum_{k=1}^N k = \frac{N}{2}(N + 1) \quad (2)$$

N = Anzahl Kriterien

Die Summe aller individuellen Vergleiche wird anschließend in einer Vergleichsmatrix zusammengetragen, wie in der nachfolgenden Abb. 20 dargestellt. Für jedes Kriterium kann nun die Gesamtsumme der gewonnenen Vergleiche errechnet werden.

Abbildung 20: Befüllte Präferenzmatrix mit Ergebnissen dreier Teilnehmer

Kriterium	K1	K2	K3	Σ	Rang R	Inverser Rang I	Relatives Gewicht G
K1		0	1	1	3	1	0,1667
	3		3	6	1	3	0,5000
K3	2	0		2	2	2	0,3333
Summe					6	1	

Quelle: Schierenbeck2016

Aus der Summe der gewonnenen Vergleiche je Kriterium wird anschließend der Rang jedes Kriteriums bestimmt. Das Kriterium, welches die meisten Paarvergleiche gewonnen hat, hat den höchsten Rang 1. Das Kriterium, welches die wenigsten Paarvergleiche gewonnen hat, bekommt den niedrigsten Rang N, welcher der Anzahl der Kriterien entspricht. Aus dem Rang kann der inverse Rang ermittelt werden, wie in der nachfolgenden Gleichung (3) dargestellt ist.

$$I_j = N + 1 - R_j \quad (3)$$

I_j = Inverser Rang des j-ten Kriteriums

N = Anzahl Kriterien

R_j = Rang des j-ten Kriteriums

Aus dem inversen Rang wird anschließend das relative Gewicht jedes Kriteriums mit der nachfolgenden Gleichung (4) bestimmt.

$$g_j = \frac{I_j}{\sum_{i=1}^N I_i} \quad (4)$$

g_j = Relatives Gewicht des j-ten Kriteriums

N = Anzahl Kriterien

I_j = Inverser Rang des j-ten Kriteriums

Dadurch erhält jedes Kriterium ein relatives Gewicht, welches in der nachfolgenden Nutzwertanalyse zur Gewichtung der Zielerfüllungsgrade genutzt werden kann. Sind alle Kriterien bekannt und gewichtet, so können die Telnutzwerte der zu vergleichenden Alternativen bestimmt werden. In der nachfolgenden Abb. 21 ist die Durchführung einer Nutzwertanalyse dargestellt.

Für jede Alternative A_1 bis A_n wird der Zielertrag k_{ij} des i-ten Kriteriums der j-ten Alternative bestimmt. Der Zielerfüllungsgrad e_{ij} wandelt den ordinal messbaren Zielertrag jedes i-ten Kriteriums jeder j-ten Alternative in einen kardinal messbaren Wert um.¹⁰⁷ Die Skala der Zielerfüllungsgrade kann nach eigenem Ermessen bestimmt werden und kann beispielsweise von eins bis fünf oder eins bis zehn Punkten reichen.

Wird das Gewicht g_i jedes i-ten Kriteriums mit dem Zielerfüllungsgrad e_{ij} jedes i-ten Kriteriums jeder j-ten Alternative multipliziert, so erhält man den Telnutzwert N_{ij} jedes i-ten Kriteriums jeder j-ten Alternative. Die Summe aller Telnutzwerte einer Alternative j ergibt den Nutzwert N_j .¹⁰⁸

¹⁰⁷Bechmann1978

¹⁰⁸Bechmann1978

Abbildung 21: Rechenschema einer Nutzwertanalyse

Alternativen A _j		A ₁			A ₂		
Kriterium K _i	Gewicht g _i	Zielertrag k _{i1}	Ziel-erfüllungsgrad e _{i1}	Teil-nutzwert N _{i1}	Zielertrag k _{i2}	Ziel-erfüllungsgrad e _{i2}	Teil-nutzwert N _{i2}
K ₁	g ₁	k ₁₁	e ₁₁	N ₁₁ =g ₁ e ₁₁	k ₁₂	e ₁₂	N ₁₂ =g ₁ e ₁₂
K ₂	g ₂	k ₂₁	e ₂₁	N ₂₁ =g ₂ e ₂₁	k ₂₂	e ₂₂	N ₂₂ =g ₂ e ₂₂
K ₃	g ₃	k ₃₁	e ₃₁	N ₃₁ =g ₃ e ₃₁	k ₃₂	e ₃₂	N ₃₂ =g ₃ e ₃₂
Summe der Gewichtungsfaktoren	$G = \sum_{i=1}^n g_i$	Nutzwert von A ₁		$N_1 = \sum_{i=1}^n N_{i1}$	Nutzwert von A ₂		$N_2 = \sum_{i=1}^n N_{i2}$

Quelle: Bechmann1978

Wenn dieses Vorgehen für alle j Alternativen wiederholt wird, so ist die Alternative mit dem höchsten Nutzwert die am besten geeignete Alternative für die gegebene Ausgangsfragestellung.

4.2 Kriterien für verschiedene Arten der Einbindung

In diesem Abschnitt werden die Kriterien definiert, anhand derer im Abschnitt 4.4 die Arten der Einbindung von Microfrontends bewertet werden.

Ein Kriterium muss, damit es bewert- und vergleichbar ist, messbar sein. Einige Kriterien sind direkt messbar (kardinal), bspw. durch physikalische Größen. Andere Kriterien sind nicht direkt messbar (ordinal), weswegen bei diesen eine Punktevergabe stattfindet, um die ordinale Messbarkeit kardinal messbar zu machen und eine Vergabe von Zielerträgen zu ermöglichen.

Die Punkte für Kriterien werden auf einer Skala von 1 bis 5 vergeben. Ergebnisse von direkt messbaren Kriterien werden miteinander verglichen, nach Optimierungen untersucht und anschließend ebenfalls in einen Punktewert auf der Skala von 1 bis 5 Punkten umgerechnet. Ein Punkt bedeutet, dass das Kriterium nicht erfüllt wurde. Fünf Punkte bedeuten, dass das Kriterium maximal erfüllt wurde. Drei Punkte stehen für eine teilweise Erfüllung. Zwei Punkte für eine geringe und vier Punkte für eine gute Erfüllung.

Nachfolgend werden die Kriterien *Übertragene Datenmenge*, *Renderingzeit*, *Entwicklungsauwand*, *Wartbarkeit*, *Nähe am Standard*, *Verschiedene Frameworks*, *Kompatibilität*, *Autonomie*, *Unabhängigkeit der Entwicklerteams*, *Lock-In Effekt* und *Interoperabilität* erläutert.

Die Kriterien entstammen zwei Expertengesprächen, welche im Rahmen der Nutzwertanalyse durchgeführt wurden. Die Protokolle der Gespräche sind in Anhang 4 in den Expertengesprächen 1 und 2 nachzulesen.

Übertragene Datenmenge

Wenn Microfrontends eingebunden werden, müssen die zur Anzeige benötigten Dateien vom Server oder CDN zum Client geladen werden. Je nachdem, mit welcher Art der Einbindung dies geschieht, variiert die Datenmenge. Eine größere Datenmenge die geladen werden muss, bedeutet eine längere Ladezeit bis zur Darstellung des Inhaltes, was besonders bei langsamen Internetverbindungen der Nutzer zu langen initialen Ladezeiten führen kann.

Die Datenmenge der Übertragung kann in Bytes gemessen werden. Die Menge variiert durch die Technologie der Komprimierung, ob Frameworks und Bibliotheken geteilt oder gecached werden und ob die Zusammensetzung der einzelnen Microfrontends serverseitig oder am Client passiert.

Die übertragene Datenmenge ist messbar, wenn der gleiche Inhalt eines Microfrontends bei mehreren Arten der Einbindung verglichen wird. Abzugrenzen ist die Datenmenge, welche der Portalshell zugehörig ist und nicht mit der Art der Einbindung in Verbindung gebracht wird. Die Datenmenge ist besonders relevant, wenn mehrere gleiche Microfrontends nebeneinander eingebunden werden. Im besten Fall skaliert die Datenmenge nicht linear mit den eingebundenen Microfrontends mit, sondern bleibt gleich.

Renderingzeit

Die Renderingzeit beschreibt die Zeit, welche vom Aufrufen durch die Portalshell bis zur Anzeige der Microfrontends vergeht. Sie hängt unter anderem mit der übertragenen Datenmenge zusammen, ist aber ebenfalls von der verwendeten Technologie abhängig. Die im Hintergrund ablaufenden Prozesse der Einbindungsarten benötigen unterschiedlich viel Zeit bis zur Darstellung des Inhaltes. Es wird in den nachfolgenden Beispielen in *Time to Usability* gemessen, also bis der gesamte Inhalt geladen und dargestellt wurde.

Andere Messverfahren, wie *Time To First Byte (TTFB)* oder *Time To First Draw (TTFD)*, wurden im Rahmen der Thesis nicht betrachtet.

Damit eine Vergleichbarkeit gegeben ist, muss auch bei diesem Kriterium der Vergleich anhand des selben Inhalts in verschiedenen Arten der Einbindung getätigt werden. Die Messung der Renderingzeit findet in Millisekunden statt.

Entwicklungsauwand

Der Entwicklungsaufwand beschreibt den nötigen Anpassungsbedarf an der Portalshell und den jeweiligen Microfrontends. Im besten Fall bedarf es keiner Anpassung auf beiden Seiten. Es fällt lediglich der Aufwand der einmaligen Einbindung des Microfrontends an. Weitere Einbindungen der gleichen Art verursachen keinen weiteren Aufwand.

Die Summe der Anpassungsaufwände, die an der Portalshell und den Microfrontends entstehen sowie die Aufwände für das einmalige Einbinden und mehrfache Einbinden werden addiert. Die Aufwände werden in Personentagen geschätzt und sind dadurch kardinal messbar.

Wartbarkeit

Die Wartbarkeit beschreibt den Aufwand, welcher erbracht werden muss, damit die Art der Einbindung auf Dauer weiterhin funktioniert. Aspekte, wie Häufigkeit von Updates oder nötige Konfigurationsänderungen, werden in der Bewertung berücksichtigt. Besonders bei hohem Entwicklungsaufwand ist Wartbarkeit wichtig. Ein langfristig geringerer Wartungsaufwand rechtfertigt einen initial höheren Entwicklungsaufwand. Die Wartbarkeit ist nicht direkt messbar, kann aber durch gesammelte Aspekte in Punkte umgerechnet werden.

Nähe am Standard

Wenn eine Lösung zur Einbindung eines Microfrontends nah am Standard der verwendeten Technologie ist, sind dadurch Risiken, wie bspw. Breaking Changes oder Empfehlung zu einer anderen Lösung unwahrscheinlich. Da die Portalapplikation auf Angular beruht, sollten die Arten der Einbindung Unterstützung durch und zugesicherte Kompatibilität mit dem Angular Framework haben. Wenn das Angular Entwicklerteam Empfehlungen aussprechen würde, eine Art der Einbindung bewusst nicht einzusetzen, würde dies zu Anpassungsaufwänden führen, weil die betroffenen Microfrontends umgebaut werden müssten.

Die Nähe zum Standard ist nicht direkt messbar, kann aber durch Vergleiche der jeweiligen Microfrontend Technologien untereinander und anhand bestehender Statements der Technologiehersteller der Portalshell verglichen werden.

Verschiedene Frameworks

Die Unterstützung verschiedener Web Frameworks bedeutet mehr Flexibilität und Autonomie der Entwicklerteams. Die Angular Portalshell aus dem vorherigen Kapitel wird nicht auf ein anderes Framework migriert.

Die Microfrontends verwenden jeweils nur ein Web Framework, diese können sich aber untereinander unterscheiden. Die Entwicklerteams müssen sich im besten Fall gar nicht gegenseitig abstimmen. Eine direkte Messung ist nicht möglich, allerdings kann eine Tendenz ermittelt werden und Abstufungen möglicher Technologien in verschiedenen Konstellationen festgehalten werden.

Kompatibilität

Die Kompatibilität beschreibt, wie gut sich die Art der Einbindung mit anderen Standards und Techniken kombinieren lässt. Eventuell müssen durch die Auswahl einer bestimmten Art der Einbindung einige Features der Portalshell oder der Microfrontends aus- oder umgebaut werden. Eine Liste an sich ausschließenden Verwendungen kann zur Bewertung herangezogen werden. Beispiele hierfür können Designframeworks sein, die miteinander nicht kompatibel sind oder es sind Features wie dynamisches Einbinden oder asynchrones Laden.

Autonomie

Die Autonomie beschreibt, inwieweit das Microfrontend und die Portalshell unabhängig voneinander existieren können. Im besten Fall sind sowohl Microfrontend, als auch die Portalshell, unabhängig voneinander. Im schlechtesten Fall kann das Microfrontend nicht ohne Parameter oder geteilte Bereiche der Portalshell funktionieren. Zur Robustheit sollten die Microfrontends autonom funktionieren.

Auch die Autonomie ist nicht direkt messbar, kann aber ebenfalls durch begründete Aspekte in Punkte umgerechnet werden.

Unabhängigkeit der Entwicklerteams

Die Unabhängigkeit der Entwicklerteams beschreibt, wie sehr die Teams, welche an verschiedenen Microfrontends arbeiten, voneinander losgelöst agieren können. Im besten Fall kann jedes Team seine eigenen Entscheidungen treffen und es treten keine Seiteneffekte auf.

Im schlechtesten Fall müssen jegliche Änderungen mit den anderen Entwicklerteams abgesprochen werden und eine autonome Arbeitsweise sowie individuelle Entscheidungsfindungen sind nicht möglich. So müssen sich eventuell Teams untereinander abstimmen, wie sie die gegenseitige Beeinflussung durch Styles und Performance unterbinden.

Dieses Kriterium ist nicht direkt messbar und muss ebenfalls in Punkte umgerechnet werden.

Lock-In Effekt

Der Lock-In Effekt beschreibt, wie stark man sich durch Verwendung dieser Art der Einbindung festsetzt. Wenn der initiale Aufwand zum Einrichten der Einbindungsart sehr hoch ist und anschließend der Aufwand zum Wechseln auf eine andere Art ebenfalls sehr hoch ist, besteht ein Lock-In Effekt. Der Lock-In Effekt ist nicht gegeben, wenn durch geringen Aufwand die Art der Einbindung gewechselt werden kann.

Es wird sowohl der Aufwand an der Portalshell, als auch der Aufwand an den Microfrontends bewertet. Ein Lock-In Effekt ist nicht klar definiert und kann daher in Abstufungen vorliegen, welche nicht kardinal messbar sind.

Interoperabilität

Die Interoperabilität beschreibt, wie gut Microfrontends durch die Art der Einbindung mit der Portalshell, aber auch anderen Microfrontends interagieren, und kommunizieren können. Zur Interoperabilität zählt das generelle Einbinden und Betreiben dieser Art. Dazu kommen Aspekte, welche sich auf Parameter- und Datenaustausch beziehen.

Bewertet werden können der Aufwand, welcher zum Einrichten von Datenübertragung nötig ist, welche Datentypen übertragen werden können, sowie die Flexibilität, welche die Schnittstellen aufbringen. Ebenfalls gibt es potentielle Konzepte der Kommunikation von Microfrontends untereinander.

Hohe Punktzahl erreichen Lösungen, die eine hohe Interoperabilität aufweisen, welche sich durch einfache, sichere und schnelle Kommunikation mit der Portalshell sowie anderen Microfrontends äußert.

Zusammenfassung

In der nachfolgenden Tabelle 3 sind alle elf Kriterien noch einmal aufgelistet. Die Kriteri-

en sind entweder objektiv messbar anhand von Einheiten oder subjektiver Natur, welche aber durch Begründungen und Nachweise belegt werden können.

Tabelle 3: Vergleichskriterien für Arten der Einbindung von Microfrontends

Nr	Kriterium	Messbar durch
1	Übertragene Datenmenge	Differenz Bytes
2	Renderingzeit	Millisekunden
3	Entwicklungsaufwand	Personentage
4	Wartbarkeit	Subjektiv
5	Nähe am Standard	Subjektiv
6	Verschiedene Frameworks	Subjektiv
7	Kompatibilität	Subjektiv
8	Autonomie	Subjektiv
9	Unabhängigkeit der Entwicklerteams	Subjektiv
10	Lock-In Effekt	Subjektiv
11	Interoperabilität	Subjektiv

Quelle: Eigene Darstellung

Die elf Kriterien der vorherigen Tabelle werden im nachfolgenden Abschnitt 4.3 geneinander nach Wichtigkeit sortiert und mit einem entsprechendem Gewicht versehen.

4.3 Gewichtung der Kriterien

In diesem Abschnitt wird die Gewichtung der Kriterien, welche im vorherigen Abschnitt 4.2 aufgelistet wurden, nach dem Prozess der in Abschnitt 4.1 erklärten Paarvergleichsmethode vorgenommen.

Das Befüllen der Paarvergleichsmatrix wurde im Rahmen eines Expertengespräches mit fünf Teilnehmern durchgeführt. Die Teilnehmer haben seit mehreren Jahren Erfahrung im Umgang mit Portalapplikationen und setzen sich zusammen aus zwei Software-Architekten, zwei Software-Entwicklern und einem IT-Projektmanager.

Im Prozess des Befüllens der Präferenzmatrix haben alle Teilnehmer individuell jedes Kriterium miteinander verglichen und dem wichtigeren Kriterium jeweils einen Punkt gegeben.

In der nachfolgenden Abb. 22 ist die ausgefüllte Paarvergleichsmatrix dargestellt, welche die Summe der fünf Einzelbewertungen enthält.

Abbildung 22: Ausgefüllte Paarvergleichsmatrix

Paarvergleichsmethode		Kriterium	Datenmenge	Renderingzeit	Entwicklungsaufwand	Wartbarkeit	Standard	Frameworks	Kompatibilität	Autonomie	Unabh. Entwicklerteams	Lock-In Effekt	Interoperabilität	Rang	Inverser Rang I	Relatives Gewicht G
Kriterium		1	2	3	4	5	6	7	8	9	10	11	Σ	R		
Datenmenge	1	0	1	0	0	0	2	1	0	0	1	0	5	11	1	0,0152
Renderingzeit	2	5	3	0	1	1	1	1	1	0	2	0	14	9	3	0,0455
Entwicklungsaufwand	3	4	2	1	2	2	2	2	2	1	4	2	22	8	4	0,0606
Wartbarkeit	4	5	5	4	5	3	4	3	3	3	4	2	38	2	10	0,1515
Standard	5	5	4	3	0		4	2	2	0	5	0	25	6	6	0,0909
Frameworks	6	3	4	3	2	1		2	2	1	4	1	23	7	5	0,0758
Kompatibilität	7	4	4	3	1	3	3		4	4	5	2	33	4	8	0,1212
Autonomie	8	5	4	3	2	3	3	1		2	4	2	29	5	7	0,1061
Unabh. Entwicklerteams	9	5	5	4	2	5	4	1	3		5	1	35	3	9	0,1364
Lock-In Effekt	10	4	3	1	1	0	1	0	1	0		1	12	10	2	0,0303
Interoperabilität	11	5	5	3	3	5	4	3	3	4	4		39	1	11	0,1667
													Summe	66	1	

Quelle: Eigene Darstellung

In der Tabelle ist zu sehen, dass das Kriterium *Interoperabilität* in Summe 39 Punkte erhalten hat und damit knapp vor *Wartbarkeit* mit 38 Punkten und *Unabhängige Entwicklerteams* mit 35 Punkten liegt. Die Spalte Rang stellt die Rangfolge der Kriterien absteigend nach ihrer bewerteten Wichtigkeit dar. Anhand des Ranges konnte mit der Gleichung (3) der inverse Rang I gebildet werden. Aus dem inversen Rang wurde für jedes Kriterium durch die Gleichung (4) das relative Gewicht G bestimmt.

Das wichtigste Kriterium *Interoperabilität* hat somit ein relatives Gewicht von 16,67% und das von den elf Kriterien am wenigsten wichtig bewertete Kriterium *Datenmenge* wird nur mit 1,52% gewichtet.

Mit den ermittelten Kriteriengewichten kann nun im nachfolgenden Schritt der Teilnutzwert jedes Kriteriums von jeder Art der Einbindung bestimmt werden.

4.4 Analyse verschiedener Integrationsansätze

In diesem Abschnitt werden die zur Auswahl stehenden alternativen Arten der Einbindung von Microfrontends in eine Portalshell verglichen. Im nachfolgenden Abschnitt 4.4.1 werden dafür zunächst die nicht in Betracht gezogenen Alternativen begründet und abgegrenzt. In den darauffolgenden Abschnitten werden anschließend die zum Vergleich ausgewählten Alternativen im Detail für jedes Kriterium untersucht und die jeweilige Erfüllung festgehalten.

4.4.1 Abgrenzung unpassender Ansätze

Zu Beginn dieser Arbeit wurden in Abschnitt 2.4, Abschnitt 2.4.1 und Abschnitt 2.5 verschiedene Arten der Realisierung von Microfrontends und Portalapplikationen beschrieben.

Portalapplikationen bestehend aus Hyperlinks sind eine einfache Möglichkeit, eine Microfrontend-Architektur darzustellen. Zur Realisierung benötigt es lediglich eine einfache Webseite, welche zu jedem referenzierten Microfrontend einen Link enthält. Bezogen auf die Beispielapplikation aus dem vorherigen Kapitel 3 können die Anforderungen in Bezug auf einheitliches Design und eingebundene, kaum spürbare Navigation nicht erfüllt werden. Durch das Klicken auf den Hyperlink wird entweder die gesamte Webseite gewechselt oder die angeklickte Seite öffnet sich in einem neuen Tab.¹⁰⁹ Dieses Verhalten steht der Anforderung von softem Routing diametral entgegen.

Ebenfalls ist das Übertragen von Parametern von der Portalapplikation zu den Microfrontends erschwert sowie eine Zugriffsbeschränkung auf Teilbereiche ebenfalls nur durch hohen Mehraufwand realisierbar.

Microfrontends durch serverseitiges Rendering sind ebenfalls keine passende Lösung, weil im prototypischen Beispiel eine Angular Portalapplikation gewählt wurde. Angular ist ein Frontend Framework für SPAs zum clientseitigen Rendern, welches softes Routing ermöglicht.

In den nachfolgenden vier Abschnitten werden die vier grundlegend zutreffenden Ansätze *Iframe*, *Web Components*, *Module Federation* und *Module Federation mit Web Components* auf die Erfüllung der in Abschnitt 4.2 definierten Kriterien untersucht.

¹⁰⁹ Steyer 2018

Damit die Kriterienerfüllung vergleichbar ist, wurde für die vier Arten der Einbindung jeweils ein identisches Microfrontend erstellt und in eine prototypische Portalshell eingebunden.

Dieses vergleichbare Microfrontend ist exemplarisch für das Iframe in Anhang 2 in Abb. 29 dargestellt. Für die anderen drei Arten der Einbindung wurde ein gleiches Microfrontend erstellt. Lediglich der Name der Einbindungsart in der Überschrift wurde entsprechend ausgetauscht. Die sich dadurch minimal unterscheidende Datenmenge durch mehr oder weniger übertragene Bytes ist zu vernachlässigen.

Das vergleichbare Microfrontend besteht aus einer Überschrift gestyled mit CSS, einem Beispielbild sowie einer Angular Material Komponente, konkret einem animierten Fortschrittsbalken.

4.4.2 Iframe

In diesem Abschnitt wird das Iframe als Microfrontend zur Einbindung in eine Portalshell bewertet. Für die Bewertung werden die im vorherigen Abschnitt 4.3 gewichteten Kriterien verwendet. Die nachfolgenden elf Kriterien sind absteigend nach ermittelten zugehörigen Gewichten sortiert.

Bei den nachfolgend kardinal messbaren Kriterien *Entwicklungsauwand*, *Renderingzeit* und *Übertragene Datenmenge* werden in diesem Abschnitt bereits die Messergebnisse der anderen drei Einbindungsarten vorweggenommen. Dadurch steht der ermittelte Zielerfüllungsgrad der Alternativen in direktem Vergleich zueinander. Die Begründung oder die Nachweise zum jeweiligen Zielertrag sind in den Abschnitten der Einbindungsarten zu finden.

Interoperabilität

Interoperabilität ist der größte Nachteil von Iframes. Der Datenaustausch zur Portalshell und zwischen den Microfrontends ist komplex.

Das Einbinden in die Portalshell ist einfach, solange keine Parameter übertragen werden müssen. Einfache Parameter können über den Querystring von der Portalshell in das Iframe Microfrontend übergeben werden. Dies funktioniert allerdings nur bei nicht sicherheitskritischen Parametern, wie bspw. dem Standort-Parameter einer Wetter-App, denn

Parameter über einen Querystring zu übertragen ist nicht sicher und kann manipuliert werden.¹¹⁰

Wenn allerdings sicherheitskritische Parameter, die entweder persönliche Informationen enthalten oder nicht modifiziert werden dürfen, vorliegen, dann ist der Querystring nicht geeignet. Dies ist dadurch bedingt, dass der Querystring unter anderem von Administratoren geloggt, aber auch vom Anwender manipuliert werden kann.

Im Fall von informationskritischen Parametern oder benötigter Authentifizierung muss eine sicherere Möglichkeit gefunden werden, um die Parameter zu überreichen und den Anwender zu authentifizieren. Dies kann über gängige Authentifizierungsprotokolle, wie OpenID Connect (OIDC) oder Security Assertion Markup Language (SAML) abgebildet werden.

Im Fall von SAML müsste eine Schnittstelle der Portalshell die zu übertragenden Informationen in einem SAML-Token verschlüsseln und dann an eine am Microfrontend konfigurierte SAML-Schnittstelle als POST-Request übertragen. Die Einbindung einer SAML-Schnittstelle kann über eine vom Portalteam bereitgestellte Bibliothek geschehen, was sich aber wiederum negativ auf den initialen Entwicklungsaufwand sowie die Autonomie der Entwicklerteams auswirkt.

Wenn ein Benutzer mehrfach den gleichen Inhalt eines Iframe Microfrontends einbinden möchte, so muss er dies auch mehrfach tun. Iframes skalieren nicht in der Einbindung und teilen auch keine Bibliotheken untereinander oder mit der Portalshell. So kommt es auch zu mehrfachen Requests und die übertragene Datenmenge steigt ebenfalls mit der Anzahl der eingebundenen gleichen IFrames.

Aus diesen Gründen wird das Kriterium *Interoperabilität* mit einer Gewichtung von 16,67% mit 1 von 5 Punkten bewertet und erhält daher einen Teilnutzwert von 0,1667.

Wartbarkeit

Der Aufwand für Wartung bei einem Iframe ist überschaubar. Im Falle einer neuen Version des Microfrontends muss dieses neu veröffentlicht werden, was allerdings durch eine einmalig eingerichtete CI/CD Pipeline automatisiert geschehen kann.

Iframes sind unabhängig von den benutzten Frontend-Frameworks und -Technologien. Von daher sind keine regelmäßigen Updates zu erwarten, auch nicht, wenn sich die

¹¹⁰Mezzalira2021

Frameworkversion beispielsweise durch ein Update von Angular 12 auf 13 erhöht. Lediglich bei einem Update der HTML-Version könnten sich theoretisch Eigenschaften des Iframe-Tags verändern, was aber eher unwahrscheinlich ist.

Das Iframe referenziert ein gehostetes Microfrontend. Wenn eine neue Version des Microfrontends gleichzeitig zu einer Bestandsversion existieren soll, müssten zwei unterschiedliche Versionen des gleichen Microfrontends auf unterschiedlichen URLs gehostet werden. Dies kann beispielsweise bei einem Rollout der Fall sein, bei dem nur konfigurierte Betatester die neuste Version erhalten und Nicht-Betatester die Bestandsversion weiterhin verwenden.

Dies kann beispielsweise über verschiedene URL-Routen der gleichen Domain realisiert werden. Das Portal muss in dem Fall wissen, für welche User es auf die jeweilige URL weiterleiten muss. Ein ähnlicher Ansatz mit dem gleichen Ziel wäre das sogenannte *Blue-Green Deployment*, bei welchem zwei Versionen des Microfrontends gleichzeitig gehostet werden und diese bei Bedarf die genutzte Route wechseln können.¹¹¹

Ein weiterer negativer Aspekt ist, dass das ganze Microfrontend im Falle einer neuen Version neu veröffentlicht werden muss, was mit einer Ausfallzeit einhergeht.

Die *Wartbarkeit* mit einer Gewichtung von 15,15% wird mit 3 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4545.

Unabhängige Entwicklerteams

Unabhängige Entwicklerteams können durch Iframes realisiert werden. Das Team, welches das Microfrontend betreut, ist grundsätzlich nicht auf andere Teams angewiesen. Es sei denn, man stimmt sich gegenseitig über Schnittstellen ab oder muss den Austausch von sicheren Parametern und Authentifizierung ermöglichen, wie vorher beim Kriterium *Interoperabilität* beschrieben. In dem Fall müssen die zu übertragenen Parameter abgestimmt werden und eine Bibliothek des Portalteams eingebunden werden, die das Auslesen der Parameter übernimmt und die Authentifizierung durchführt.

Wenn das Microfrontendteam nicht in der gleichen Programmiersprache, wie das Portalteam entwickelt, muss das Portalteam eine Schnittstellendokumentation zur Verfügung

¹¹¹Mehr Informationen über Blue-Green Deployment unter **CloudFoundryDocumentation2020**

stellen, anhand derer die Funktionen nachgebaut und die Parameter ausgelesen werden können.

Das Kriterium *Unabhängige Entwicklerteams* mit einer Gewichtung von 13,64% wird mit 4 von 5 Punkten bewertet, weil lediglich Abhängigkeit zwischen dem Portalteam und dem Microfrontendteam besteht, wenn Authentifizierung oder sicherer Parameteraustausch durchgeführt werden muss. Der Teilnutzwert des Kriteriums beträgt demnach 0,5456.

Kompatibilität

Das Iframe wird vollständig mit allem Seitenmarkup in dem DOM-Tree der Portalapplikation eingebunden. Dadurch doppeln sich HTML Elemente, wie zum Beispiel der Header. Screenreader und Crawler von Suchmaschinen sind durch diese Seitenstruktur häufig verwirrt und erzielen schlechte Resultate.¹¹²

Ebenfalls weist das Iframe diverse Sicherheitslücken auf, die ausgenutzt werden können. So sind *Iframe Phishing*, *Clickjacking*, *Cross-Frame Scripting* und *Iframe Injection* Sicherheitsrisiken, welche Hacker auf Webseiten mit Iframes ausnutzen können.¹¹³

Positiv gilt es zu erwähnen, dass das Iframe als einzige Art der Einbindung für veraltete Browser zugänglich ist und sogar im Internet Explorer seit jeher verwendet werden kann.¹¹⁴

Das Kriterium *Kompatibilität* mit einer Gewichtung von 12,12% wird mit einem Punkt bewertet und erhält dadurch einen Teilnutzwert von 0,1212.

Autonomie

Da der Iframe-Tag eine gehostete Webseite einbindet, läuft die eingebundene Applikation grundsätzlich autonom. Eine Applikation, welche keine mandantenspezifischen Parameter benötigt läuft dementsprechend komplett selbstständig. Eine Applikation, welche Inputparameter der Portalshell benötigt, wie beispielsweise eine Wetter-App, funktioniert ohne die benötigten Standort-Parameter nicht.

Ein weiterer Vorteil ist, dass das Iframe die Portalshell nicht beeinflussen kann. So sind die Stylings abgekapselt. Das Microfrontend und die Portalshell beeinflussen sich

¹¹²GoogleSearchCentral2022

¹¹³Gunawardhana2021

¹¹⁴MDNWebDocs2021a

nicht gegenseitig.¹¹⁵ Dies ist in Anhang 2 in Abb. 40 dargestellt. Der rot umrandete Bereich ist das Iframe, welches die Background-Color von allen Überschriften rot setzt. Alles außerhalb des roten Rahmens sind Elemente von der Portalshell und dort haben Überschriften eine blaue Hintergrundfarbe. Es findet keine gegenseitige Beeinflussung statt.

Das Kriterium *Autonomie* mit einer Gewichtung von 10,61% wird mit 5 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,5305. Nur verlinkte Portalapplikationen haben eine noch höhere Autonomie.

Nähe zum Standard

Das Iframe ist ein offiziell anerkanntes HTML Element, welches bereits seit vielen Jahren besteht. Alle Browser unterstützen das Iframe mit den grundlegenden Parametern `src`, `height` und `width`.¹¹⁶ Alle Frontend Frameworks, welche HTML einsetzen, können Iframes einbinden.

Das Kriterium *Nähe zum Standard* mit einer Gewichtung von 9,09% wird mit 5 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4545.

Verschiedene Frameworks

Mit einem Iframe gibt es kaum Beschränkungen und Inkompatibilitäten zu anderen Frameworks. Da das Iframe durch einen HTML-Tag eingebunden wird, ist es mit allen Frontend-Frameworks kompatibel, welche HTML nutzen. Weniger Beschränkungen gäbe es nur noch bei reinen Applikationen auf der Basis von Links.

Das Iframe bietet sich für legacy Anwendungen an, die nicht auf moderne Web-Frameworks umgebaut werden können. Die Art der Einbindung kostet wenig Budget und ist schnell umsetzbar. Auch kann der Iframe Inhalt serverseitig gerendert werden, selbst wenn die Portalshell beispielsweise clientseitig gerendert wird.

Deswegen wird das Kriterium *Frameworks* mit einer Gewichtung von 7,58% mit 5 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,3790.

Entwicklungsaufwand

In der nachfolgenden Tabelle 4 wird eine Gegenüberstellung des Entwicklungsaufwandes bei verschiedenen Arten der Einbindung vorgenommen. Alle vier Arten wurden nach initialem Entwicklungsaufwand, Aufwand der bei Änderung an Features entsteht

¹¹⁵Geers2020

¹¹⁶MDNWebDocs2021a

und Aufwand, welcher bei Einbindung des gleichen Microfrontends an einer anderen Stelle der Portalapplikation auftritt, bewertet. Letzteres findet häufig bei Portalapplikationen mit Mandantenkontext Anwendung. Verschiedene Mandanten rufen das gleiche Microfrontend mit unterschiedlichen, mandantenindividuellen Parametern auf.

Die Bewertungsskala reicht von ++ (sehr gut), über + (gut), o (neutral), - (schlecht) bis hin zu - - (sehr schlecht). Ein sehr geringer Aufwand erhält ein ++. Ein sehr hoher Aufwand bekommt eine sehr schlechte (- -) Bewertung. Der Gesamtaufwand wird in Personentage (PT) geschätzt. Ein PT entspricht 8 Stunden.

Tabelle 4: Entwicklungsaufwand verschiedener Microfrontends

Art der Einbindung	Initialer Aufwand	Anpassungsaufwand	Aufwand mehrfache Einbindung	Geschätzter Aufwand PT
Iframe	++	-	o	0,75
Web Component	+	-	++	1
Module Federation	-	+	+	2
Module Federation WC	--	-	+	2,5

Quelle: Eigene Darstellung

Der initiale Entwicklungsaufwand ist bei einer Einbindung über ein Iframe gering. So muss das Microfrontend lediglich entwickelt werden und auf einer Webseite veröffentlicht sein, die das Einbinden über den *CORS Header* erlaubt.

Bei der einbindenden Portalapplikation muss der Iframe HTML-Tag konfiguriert werden. Der src-Parameter zeigt auf die URL des veröffentlichten Microfrontends. Es empfiehlt sich, das standardmäßige Styling zu überschreiben, damit die Höhe und Breite dem angezeigten Content entsprechen und der standardmäßige Rand entfernt wird. Ebenfalls sollte die Einbindung so gebaut werden, dass sie dynamisch geschieht und nicht direkt im Code referenziert wird.

Der Aufwand dafür ist gering, sodass das Iframe bereits nach wenigen Minuten angezeigt werden kann. Noch schneller ginge nur das direkte Einbinden eines Links, wie es in Abschnitt 2.4.1 beschrieben wurde. Die Einbindung des Iframes skaliert nicht, das heißt es muss jedes mal wieder ein iframe-Tag eingebunden werden. Ändern sich die Hosting-URL oder die benötigten Parameter der Iframe-Webseite, so funktioniert die Einbindung nicht mehr und muss angepasst werden.

Aus diesem Grund wird das Kriterium *Entwicklungsaufwand* bei einem Iframe mit 4 von 5 Punkten bewertet. Bei der Gewichtung des Kriteriums von 6,06% wird ein Teilnutzwert von 0,2424 erreicht.

Renderingzeit

Die Renderingzeit für Iframes ist, wie in der nachfolgenden Tabelle 5 dargestellt, mit 480ms die höchste von allen vier verglichenen Arten der Einbindung. Eine halbe Sekunde ist eine spürbare Verzögerung beim Aufrufen einer Seite. Allerdings stellt eine halbe Sekunde dennoch eine ertragbare Wartezeit beim Arbeiten mit der Webseite dar.¹¹⁷

Tabelle 5: Übersicht Datenmenge & Renderingzeit von Einbindungsarten

Art der Einbindung	Datenmenge in KB	Datenmenge mit großer Bibliothek	Renderingzeit in ms
Iframe	1949	2254	480
Web Component	1824	2205	60
Module Federation	2523	2524	39
Module Federation WC	2609	2611	23

Quelle: Eigene Darstellung

Die Daten aus der Tabelle entstammen Messungen, welche in Anhang 2 in den Abb. 30 bis 33 dargestellt sind. Die Messungen wurden mit dem gleichen Microfrontend durchgeführt, welches durch unterschiedliche Arten eingebunden wurde. Damit die Messungen nachvollzogen werden können, ist in Anhang 2 in Abb. 57 das Laborsetup mit den Messbedingungen dargestellt.

Dadurch, dass die anderen drei Arten Renderingzeiten im zweistelligen Millisekunden haben, wird das Iframe nur mit 2 von 5 Punkten bewertet. Durch die Gewichtung des Kriteriums *Renderingzeit* von 4,55% erreicht das Iframe einen Teilnutzwert von 0,0910.

Lock-In Effekt

Bei einer Einbindung durch ein Iframe entsteht kein Lock-In Effekt. Es wird weder auf Seiten des Microfrontends, noch auf Seiten der Portalshell eine proprietäre Technologie eingesetzt. Auf Seiten des Microfrontends bedarf es keiner Anpassung. Bei der Portalshell muss zum Entfernen des Iframes lediglich der einbindende HTML-Tag gelöscht werden.

¹¹⁷unbounce2022

Jede der drei nachfolgenden Arten der Einbindung kann autonom funktionieren und dadurch als Iframe eingebunden werden. Beispielsweise kann die Web Component sich selbst in der Index.html einbinden, wie exemplarisch in Listing 18 in Anhang 1 zu sehen ist. Anschließend kann die Index.html als Iframe in einer Portalapplikation eingebunden werden.

Selbiges gilt für die Einbindungen über Module Federation. Das Modul, welches standardmäßig exportiert wird, kann auch von einem Iframe eingebunden werden.

Das Kriterium *Lock-In Effekt* mit einer Gewichtung von 3,03% wird mit 5 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,1515.

Übertragene Datenmenge

In der vorherigen Tabelle 5 wurde bei den vier Arten der Einbindung jeweils die übertragene Datenmenge sowie die benötigte Renderingzeit gemessen und gegenübergestellt. Das Iframe weist mit 1949 Kilobyte (KB) im normalen Anwendungsfall und 2254 KB bei Verwendung einer großen Bibliothek verglichen mit den anderen Arten geringe Werte auf. So überträgt das Iframe, knapp hinter der Web Component, die geringste Datenmenge zum Client.

Dennoch ist die Datenmenge bei einem Iframe nicht skalierbar. Werden mehrere Bibliotheken im Iframe eingebunden, so werden diese alle zum Client übertragen. Werden mehrere Iframes auf einer Seite eingebunden, so werden alle Bibliotheken aller Iframes übertragen. Die übertragene Datenmenge skaliert linear mit der Anzahl der eingebundenen Iframes. Dies ist exemplarisch in der Abb. 39 in Anhang 2 mit zwei Iframes auf der gleichen Seite dargestellt. Für beide werden der Code sowie die Bibliotheken zum Client geladen.

Die übertragende Datenmenge setzt sich aus der generierten JS-Datei, welche die Bibliotheken beinhalten, sowie den Stylings zusammen. Bibliotheken oder Assets können nicht geteilt werden. Bei dem Iframe wird der gesamte HTML-Inhalt der eingebundenen Webseite in das HTML der Portalshell gerendert, wodurch Overhead entsteht (siehe Abb. 34 in Anhang 2).

Das Kriterium *Datenmenge* mit einer Gewichtung von 1,52% wird daher mit nur einem von fünf möglichen Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,0152.

Zwischenfazit

Das Iframe ist eine schnelle Lösung, welches sich für veraltete Anwendungen anbietet, in die nicht mehr investiert werden soll. Allerdings ist die Interoperabilität nicht ausgeprägt und besonders komplexe Einbindungen mit vielen geteilten Parametern, Sicherheitsanforderungen und bidirektionalem Austausch sind nicht geeignet. Ebenfalls sind Ladezeiten sowie Datenmenge nicht optimiert und skalieren nicht.

4.4.3 Web Components

In diesem Abschnitt wird die Web Component als Art der Einbindung in eine Portalshell untersucht. Die Web Component wird, wie das Iframe im vorherigen Abschnitt 4.4.2, auf die Erfüllung der elf gewichteten Kriterien aus Abschnitt 4.3 geprüft.

Interoperabilität

In Microfrontends durch Web Components können mittels Parametern Daten von der Portalshell entgegengenommen werden. Durch diese Parameter kann das Microfrontend individuell auf die Bedürfnisse des aktuellen Benutzers, dessen Mandanten und der Portalshell angepasst werden. Im Beispiel eines Wetter-Microfrontends könnten die Standort-Daten des Nutzers übertragen werden, sodass diesem immer das aktuelle Wetter angezeigt wird.

Muss der Nutzer im Microfrontend authentifiziert werden, ist der Prozess aufwändiger. Die Portalshell muss der Web Component einen OIDC Token übergeben, welcher vom Microfrontend genutzt wird, um einen Aufruf an das Backend der Portalshell zu tätigen. Die Portalshell verifiziert die Aufrufe des Microfrontends und gibt Daten für autorisierte Endpunkte zurück.

Die Web Component kann über Komponentenbibliotheken mit anderen Microfrontends oder der Portalshell kommunizieren und Daten austauschen. Alternativ kann die Kommunikation zwischen den Microfrontends und der Portalshell auch über Events geschehen, auf die die Empfänger reagieren. So kann beispielsweise bei einer Web Component Tabelle mit *infinite scroll*-Feature ein Event gesendet werden, sobald der Benutzer am unteren Tabellenrand angekommen ist und die Portalshell weitere Einträge liefern soll.¹¹⁸

¹¹⁸Rauber2020d

Im Gegensatz zum Iframe reicht die einmalige Einbindung einer Web Component und anschließend kann das Microfrontend mehrfach aufgerufen und eingebunden werden.

Das Kriterium *Interoperabilität* mit einer Gewichtung von 16,67% wird mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,6668.

Wartbarkeit

Der Wartungsaufwand für Web Components ist ebenfalls gering. Wenn eine Web Component einmal gebaut wurde, kann sie in alle anderen Applikationen eingebunden werden. Es existieren Onlineplattformen, welche veröffentlichte Web Components verwalteten und Einbindung dieser in andere Applikationen ermöglichen, wie bspw. <https://www.webcomponents.org/>.

Die Verwaltung von internen Web Components muss aber nicht zwingend über einen Paketmanager geschehen, sondern kann auch über einen erreichbaren Speicherort gelöst werden (bspw. einen Azure Storage Account). Im Storage liegt dann in einem Unterordner, welcher nach der Version benannt ist, die kompilierte JS-Datei. Die Portalshell kann dann nach Mitteilung des Web Component Teams eine neue Version manuell einbinden, wofür der Direktlink zur Datei ausreicht.

Das Ablegen der Dateien im Storage oder Paketmanager kann durch eine CI/CD Pipeline automatisiert werden. Auch das Abrufen sowie Einbinden der Versionen in der Portalshell kann automatisiert geschehen. Es ist zeitgleich möglich, mehrere verschiedene Versionen der gleichen Web Component einzubinden, solange der HTML-Tag sich für jede Version unterscheidet.

Das Kriterium *Wartbarkeit* mit einer Gewichtung von 15,15% wird mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,6060.

Unabhängige Entwicklerteams

Das Microfrontend Team kann bei Web Components unabhängig vom Entwicklungsteam der Portalshell agieren. Denn das Team, welches die Web Component betreut kann jederzeit eine neue Version der Web Component veröffentlichen, solange sie die alte Version nicht überschreiben.

Über Paketmanager, wie den Node Package Manager (NPM) oder Yarn, können Web Components in verschiedenen Version veröffentlicht werden. So bleibt eine Historie der

Web Components erhalten und das Portalshellteam kann je nach Bedarf die neueste oder eine ältere Version einbinden.

Vorteile eines Paketmanagers sind, dass öffentliche Web Components kostenfrei eingebunden werden können. Der Sourcecode von internen Web Components sollte privat bleiben, was aber nur durch Bezahlmodelle gelöst werden kann (bspw. NPM Organizations).¹¹⁹ Eine eigens kreierte Lösung ist dahingegen kostenfrei, allerdings nicht vergleichbar flexibel mit Versionen und der automatischen Einbindung.

Aus dem Grund, dass Web Components Kompatibilitätsprobleme mit einigen Designframeworks haben, sind die Teams in ihrer Entscheidungsauswahl eingeschränkt. Eventuell müssen Bestandslösungen angepasst oder Neuentwicklungen mit anderen Design-Komponenten gebaut werden.

Das Kriterium *Unabhängige Entwicklerteams* mit einer Gewichtung von 13,64% wird für Web Components daher mit 3 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4092.

Kompatibilität

Web Components basieren auf den Grundlagen von JS sowie HTML und sind daher unabhängig von Frontend-Frameworks einsetzbar.

Sie werden von den modernen Browsern wie Chrome, Edge, Firefox und Opera vollumfänglich unterstützt. Safari unterstützt einige der Web Components Features, aber nicht alle. Der Internet Explorer kann keine Web Components darstellen.¹²⁰ Der Internet Explorer wird ab Mitte 2022 offiziell nicht mehr von Microsoft unterstützt.¹²¹ Er wurde weitestgehend durch den Edge-Browser ersetzt und ist im Jahre 2022 nur noch bei weniger als einem Prozent der deutschen Nutzer im Einsatz.¹²²

Durch die Einbindung von Web Components im Shadow DOM wird verhindert, dass Stylings der Komponente die Portalshell beeinflussen. Allerdings entstehen dadurch Probleme mit Design Frameworks, die auf globale Styles setzen. Bei Angular Material¹²³ oder Twitter Bootstrap¹²⁴ kann es deshalb zu Problemen bei der Darstellung kommen.

¹¹⁹**npm2022**

¹²⁰**MDNWebDocs2022a**

¹²¹**Microsoft2021b**

¹²²Vgl. Abb. 36 in Anhang 2

¹²³**Github2021**

¹²⁴**Geers2020**

Dies kann aber durch das Einbinden der Styles in der Komponente oder durch das Entfernen der Shadow DOM Einbindung gelöst werden. Weitere Informationen zu Shadow DOM sind im nachfolgenden Kriterium *Autonomie* beschrieben.

Das Kriterium *Kompatibilität* mit einer Gewichtung von 12,12% wird daher mit 3 von 5 Punkten bewertet und erhält somit einen Teilnutzwert von 0,3636.

Autonomie

Angular Komponenten, welche zu einer Web Component definiert werden, haben verschiedene Ausprägungen der sogenannten *encapsulation policy*, welche die Reichweite des Stylings der Komponente steuert. Die Ausprägung `ViewEncapsulation.None` bedeutet, dass keine Kapselung des Stylings geschieht und die Stylings auch andere Komponenten betreffen könnten. Bei `ViewEncapsulation.Emulated` werden die Dateien der Web Component an einen emulierten, gekapselten DOM-Tree des Browsers angehangen und wirken nur dort beschränkt. Durch `ViewEncapsulation.ShadowDom` wird die Shadow DOM API des Browsers genutzt und die Stylings dadurch beschränkt.¹²⁵

Die Stylings des Microfrontends können andere Elemente der Portalshell im Modus `ViewEncapsulation.ShadowDom` nicht beeinflussen. Allerdings können dadurch Probleme mit dem Styling einiger eingebundener Komponenten in der Web Component entstehen, wie beispielsweise die fehlerhafte Darstellung von Angular Material Komponenten.¹²⁶

Das Kriterium *Autonomie* mit einer Gewichtung von 10,61% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4244.

Nähe zum Standard

Web Components und die dazugehörigen Features, wie `customElements`, `HTML Templates` und `Shadow DOM`, sind Standards der Web Entwicklung.¹²⁷

Von daher sind Web Components eine Lösung, welche dem Standard entspricht. Das Kriterium *Nähe zum Standard* mit einer Gewichtung von 9,09% wird dementsprechend mit 5 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4545.

¹²⁵Angular2022a

¹²⁶Handler2020

¹²⁷MDNWebDocs2022a

Verschiedene Frameworks

Die Webseite <https://custom-elements-everywhere.com/> prüft diverse Frontend Frameworks bei jedem Update automatisiert auf die Kompatibilität zur customElements-API.¹²⁸

Zum Zeitpunkt der Erstellung werden unter anderem *Vue.js*, *React*, *Angular*, *AngularJS* und *Svelte* mit positivem Testergebnis ausgewiesen. Die relevanten Frameworks zur Erstellung von Web Components sind dadurch kompatibel und decken sich ebenfalls mit vielen der populärsten Web Frameworks (vgl. Abb. 35 in Anhang 2).

Das Kriterium *Frameworks* mit einer Gewichtung von 7,58% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,3032.

Entwicklusaufwand

in Anhang 1 in Listing 8 und Listing 15 ist zu sehen, welche Codeanpassungen zum Einbinden einer Web Component auf Seiten der Portalshell und des Microfrontends nötig sind. Bei dem Microfrontend muss in der `app.module.ts` die Angular Komponente, welche exportiert werden soll, als `customElement` definiert werden. Wird das gleiche Element zweifach definiert, kommt es zu Fehlern, weswegen vorher geprüft wird, ob bereits ein Element mit dem Namen besteht. Anschließend kann das Microfrontend gebaut und die `main.js` in die Portalshell eingebunden werden.

Damit die Portalshell eine Web Component einbinden kann, müssen drei Aspekte bekannt sein. Die URL der `main.js`, der Name des *customElements* und ob Parameter mit übergeben werden sollen. Die Portalshell muss eine Möglichkeit zur Verfügung stellen, welche die Einbindung der Web Component übernimmt (bspw. eine Angular Komponente).

Die Komponente der Portalshell muss den JS-Code laden, ein neues Script-Tag in das HTML-Dokument der Portalshell einfügen und die URL der Web Component auf die `src` Property des Script-Tags setzen, wie in Anhang 1 unter Listing 15 zu sehen ist. Anschließend wird der `createElement`-Methode der Name des definierten HTML-Elements übergeben und dadurch die Web Component eingebunden.

Der Aufwand ist etwas höher als bei einem Iframe, aber in Summe dennoch überschaubar. Wenn die Methoden zum Einbinden entwickelt wurden, können sie gut wiederverwendet werden. Das Microfrontend kann nach einmaliger Konfiguration mehrfach eingebunden

¹²⁸CustomElementsEverywhere2022

werden, bei Bedarf auch mit unterschiedlichen Parametern. In Tabelle 4 wurde deswegen die Web Component mit einer guten Wertung beim initialen Aufwand und einer sehr guten Wertung beim mehrfachen Verwenden versehen. Die schlechte Wertung beim Anpassungsaufwand bedingt sich durch den Anpassungsbedarf, wenn Parameter oder die URL der JS-Datei sich ändern.

Das Kriterium *Entwicklungs aufwand* mit einer Gewichtung von 6,06% wird mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,2424.

Renderingzeit

In der im vorherigen Abschnitt aufgestellten Tabelle 5 weist die Web Component als Art der Einbindung eine Renderingzeit von 60ms auf. Damit ist diese Art zwar nur die Drittschnellste, aber eine Renderingzeit im zweistelligen Millisekundenbereich ist ausreichend schnell und stört den Nutzer nicht merklich.

Dadurch wird das Kriterium *Renderingzeit* mit 3 von 5 Punkten bewertet und erreicht bei einer Gewichtung von 4,55% einen Teilnutzwert von 0,1365.

Lock-In Effekt

Ein Lock-In Effekt liegt bei Web Components nicht vor. Die Web Component ist, wie bereits beim Kriterium *Nähe zum Standard* erläutert, ein anerkannter Standard von HTML und JS. Sollte man sich in Zukunft gegen Web Components entscheiden, kann das Microfrontend der Web Component auch ohne nennenswerten Mehraufwand direkt als Iframe eingebunden werden.

In Anhang 1 in Listing 18 ist der nötige Code zu sehen, dessen es bedarf, damit ein Web Component Microfrontend sich selbst einbindet. Es reicht den vorher definierten HTML-Tag in die `index.html` einzufügen. Das Ergebnis der Selbsteinbindung ist in Anhang 2 unter Abb. 48 dargestellt. Nach dieser Einbindung kann die Webseite des Microfrontends veröffentlicht und in andere Applikationen als Iframe eingebunden werden. Es entsteht daher kein Lock-In Effekt.

Deswegen erhält *Lock-In Effekt* 5 von 5 möglichen Punkten, was bei der ermittelten Gewichtung von 3,03% zu einem Teilnutzwert von 0,1515 führt.

Datenmenge

Die Datenmenge eines vergleichbaren Microfrontends hat im einheitlichen Test mit 1824KB die geringste Datenmenge der vier verglichenen Arten der Einbindung erreicht (siehe Tabelle 5). Wurde eine große Bibliothek in der Web Component verwendet, so

werden 2205KB zum Client übertragen. Die Skalierung der übertragenen Dateigröße gleicht der beim Iframe.

Ist die gleiche Web Component in mehreren Microfrontends in der Portalshell eingebunden, steigt die Datenmenge nicht linear zur Anzahl der Microfrontends an, wie es beim Iframe der Fall ist. Die Web Component wird clientseitig durch den Browser gecached und muss somit nur einmalig geladen werden.¹²⁹

Im nachfolgenden Abschnitt 4.4.5 wird ein Vergleich der übertragenen Datenmenge bei einem Microfrontend mit mehreren eingebundenen Bibliotheken mit Web Components und Module Federation mit Web Components durchgeführt. Der Vergleich zeigt, dass, ab einer bestimmten Summe der Datenmenge von geteilten Bibliotheken, die Einbindung über Module Federation in einer geringeren zum Client übertragenen Datenmenge resultiert. Wird dieser Punkt jedoch nicht überschritten, so sind Web Components sinnvoller.

Das Kriterium *Datenmenge* mit einer Gewichtung von 1,52% wird mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,0608.

Zwischenfazit

In diesem Abschnitt wurden Web Components als Art der Einbindung in eine Portalshell untersucht. Die Web Components weisen Stärken bei der Skalierbarkeit, der Kompatibilität, der Datenmenge bei kleinen Bibliotheken und bei der Kapselung der Inhalte auf. Allerdings entstehen durch die Kapselung auch Probleme bei der Anzeige und bei großen eingebundenen Bibliotheken ist die übertragene Datenmenge nicht optimal.

4.4.4 Module Federation mit Angular Modulen

In diesem Abschnitt wird Module Federation mit Angular Modulen als Einbindungsart eines Microfrontends in eine Portalshell bewertet. Dies geschieht, wie bei den beiden vorherigen Abschnitten, durch die Feststellung der Erfüllung der in Abschnitt 4.3 gewichteten Kriterien.

¹²⁹Siehe Abb. 54 in Anhang 2

Interoperabilität

Module Federation teilt Daten von der Portalshell zum Microfrontend nicht über Parameter, wie es bei den beiden vorherigen Arten der Fall war, sondern über *Angular Libraries (Lib)s*. Diese beinhalten einen Service und optional weitere Logik.¹³⁰

Ein Beispiel dafür ist in Anhang 2 unter Abb. 37 dargestellt. Dieses zeigt eine Lib zur Authentifizierung, welche Informationen über den angemeldeten User von der Portalshell zum Microfrontend reicht. Der User loggt sich auf der Loginmaske mit seinen Daten ein. Die Portalshell validiert die Logindaten und speichert den Namen des Users anschließend in einem Service der Lib, auf welchen das Module Federation Microfrontend zugreifen kann.

Damit Parameter an das Module Federation Microfrontend übergeben werden können, sollte dafür also eine Lib erstellt werden. Die Lib muss vom Portalteam verwaltet sowie unter Umständen erweitert werden können. Sie beinhaltet die Parameter, welche für die Microfrontends notwendig sind. Eine Lib zu erstellen bedeutet mehr Aufwand, als nur Parameter zu übergeben, wie bspw. bei der Web Component. Allerdings können darüber auch komplexe Logiken abgebildet werden, wie beispielsweise eine Auth0-Implementierung zur Authentifizierung.¹³¹

Die Microfrontends, welche durch Module Federation in die Portalshell eingebunden sind, werden durch sogenanntes *lazy loading* verzögert eingebunden. Dies ist eine Ausnahme gegenüber anderen Arten der Einbindung, bei denen jeglicher Code zur Compilezeit bekannt sein muss.¹³² Die asynchrone Einbindung wirkt sich jedoch positiv auf die initiale Renderingzeit der Portalshell aus.

Das Kriterium *Interoperabilität* mit einer Gewichtung von 16,67% wird daher mit 5 von 5 Punkten bewertet und erhält somit einen Teilnutzwert von 0,8335.

Wartbarkeit

In Zukunft sind Updates für Module Federation zu erwarten, weil die Art der Einbindung erst seit Anfang 2020 und damit seit aktuell ca. zwei Jahren zur Verfügung steht.¹³³ Durch produktiven Einsatz in Unternehmenskontexten können sich in Zukunft noch neue Anforderungen ergeben. Updates der Webpack Version oder der Module Federation

¹³⁰ Steyer2020

¹³¹ Steyer2022a

¹³² Steyer2021c

¹³³ Jackson2020b

Hilfsbibliotheken sind in Zukunft zu erwarten und müssen bei allen Microfrontends sowie der Portalshell zeitgleich eingespielt werden.

In der offiziellen Webpack Dokumentation zu Module Federation sind zwei beispielhafte Anwendungsszenarien skizziert. So kann eine von mehreren anderen Microfrontends genutzte Bibliothek veröffentlicht und über Module Federation eingebunden werden. Dies können sich verteilte Teams zu Nutzen machen. Für die geteilte Bibliothek kann bei Bedarf einzeln ein Update eingespielt werden, ohne dass alle abhängigen Microfrontends ebenfalls neu veröffentlicht werden müssen.¹³⁴ Dies stärkt die Wartbarkeit sowie die Autonomie der zugehörigen Microfrontends.

Dadurch erhält das Kriterium *Wartbarkeit* 4 von 5 möglichen Punkten, was bei der Gewichtung von 15,15% zu einem Teilnutzwert von 0,6060 führt.

Unabhängige Entwicklerteams

Das Entwicklerteam des Microfrontends kann unabhängig von Entwicklerteams anderer Microfrontends und der Portalshell agieren und neue Versionen veröffentlichen. Allerdings sind zur Verminderung der Datenmenge einige Bibliotheken mit der Portalshell geteilt. Im konkreten Fall der vorher erwähnten Beispielapplikation, welche in Listing 7 in Anhang 1 dargestellt ist, sind dies die Angular-Bibliotheken `@angular/core`, `@angular/common`, `@angular/router` und `@angular/material`.

Das Portalteam muss sich mit den Microfrontendteams abstimmen, welche Versionen sie nutzen. Ein unterschiedliche Major Version des Angular Core Frameworks kann unter Umständen die Funktionalität des Microfrontends beeinträchtigen. Wenn ein Versionsunterschied nicht zu vermeiden ist, dann muss die Funktionalität durch eine hohe Testabdeckung gewährleistet werden.¹³⁵

Alternativ kann auf das Teilen der Bibliotheken verzichtet werden. Hierdurch kann das Microfrontend eine andere Version der Bibliothek verwenden als die Portalshell. Dies wirkt sich dann allerdings negativ auf die übertragene Datenmenge aus. Die nötige Abstimmung der Teams untereinander reduziert die Unabhängigkeit, ist aber auch keine zwingende Voraussetzung. Es muss abgewogen werden, ob die reduzierte Datenmenge den Abstimmungsaufwand rechtfertigt.

¹³⁴Webpack2020b

¹³⁵Steyer2021a

Da Module Federation auf Webpack beruht, sind die Microfrontendteams in der Entscheidung des Web Frontend Frameworks eingeschränkt. Die meisten Web Frontend Frameworks beruhen zwar auf Webpack, allerdings wäre die Einbindung eines Microfrontends ohne Webpack (bspw. nur bestehend aus HTML, CSS und JS) nicht möglich.

Das Kriterium *Unabhängige Entwicklerteams* mit einer Gewichtung von 13,64% wird daher mit 3 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4092.

Kompatibilität

Bei Module Federation wird ein extern kompiliertes Angular Modul in eine andere Angular Applikation eingebunden. Die einbindende Portalshell kennt das Microfrontend nicht und geht nur von einem standardmäßigen lazy loading eines Modules aus.¹³⁶

Daher sollte zur Stabilität die Angular Version des eingebundenen Microfrontends der Angular Version der Portalshell entsprechen, weil es ansonsten zu Problemen kommen könnte. Andere Web Frameworks sind nur über die Einbindung von Module Federation mit Web Components möglich, da diese nicht über Angular Module und Angular Komponenten verfügen.

Durch Module Federation können auch umfangreiche Microfrontends mit eigenem Routing eingebunden werden. Dafür muss aber der Angular Router wissen, welcher Teil der Route zur Portalshell und welcher Teil zum Microfrontend gehört. Dies kann entweder über die `startsWith`-Methode von dem NPM-Paket `@angular-architects/module-federation-tools` gelöst werden oder die Teams müssen die Funktion selber implementieren.¹³⁷

Das Kriterium *Kompatibilität* mit einer Gewichtung von 12,12% wird daher mit 3 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,3636.

Autonomie

Durch das Teilen von Bibliotheken ist das Microfrontend von der Portalshell abhängig. Wenn die Portalshell die vom Microfrontend benötigten Bibliotheken nicht vorweisen kann, kommt es zu Fehlern. Allerdings müssen die Bibliotheken nicht zwangsläufig geteilt werden, was sich dann jedoch negativ auf die übertragene Datenmenge auswirkt. Es gibt

¹³⁶Steyer2021a

¹³⁷Steyer2021a

verschiedene Konfigurationsmöglichkeiten, um die benötigten Versionen der geteilten Bibliotheken zu bestimmen. Je nach Konfiguration werden gewollt Fehler ausgegeben, sollte es keine gemeinsame geteilte Version geben oder es werden unterschiedliche Versionen in einer gewissen Spanne in Kauf genommen.¹³⁸

Ein Konzept von Module Federation sieht vor, dass alle Unterseiten einer SPA als eigenständiges Module Federation Frontend veröffentlicht werden. Dadurch können alle Seiten bei Bedarf individuell neu veröffentlicht werden, ohne dass eine Downtime der Portalshell oder anderer eingebundener Microfrontends auftritt. Die Portalshell verlinkt die anderen eingebundenen Module Federation Microfrontends und teilt Bibliotheken, damit diese nicht mehrfach gebaut und vom Client geladen werden. Die Portalshell muss lediglich bei Änderungen der Routen und weiteren geteilten Bibliotheken neu veröffentlicht werden.¹³⁹

Diese Vorgehensweise beschreibt den klassischen Microfrontend-Architektur Ansatz. Wenn die Portalshell zusätzlich auch die Microfrontend-Konfigurationen zum Einbinden dynamisch ausliest, sind alle Teams vollständig autonom. Sollte eines der Microfrontends ausfallen, sind die anderen nicht betroffen. Lediglich die Portalshell muss ausfallsicher gehostet werden, bspw. durch *Load Balancing*.

Das Kriterium *Autonomie* mit einer Gewichtung von 10,61% wird daher mit 4 von 5 Punkten bewertet und erreicht dadurch einen Teilnutzwert von 0,4244.

Nähe zum Standard

Module Federation wird zwar durch einige Google Developer Experts (GDE) verbreitet, hat aber keine offizielle Unterstützung durch Angular. Die Angular Entwickler empfehlen Module Federation zu nutzen, garantieren aber nicht für Wartbarkeit und Stabilität.¹⁴⁰ Damit entspricht Module Federation zum Stand der Bearbeitung (Q1/2022) nicht dem offiziellen Standard.

Allerdings stehen Microfrontend-Architekturen auf der Agenda des Angular Teams für zukünftige Erweiterungen. Es wird berichtet, dass im Jahr 2022 vermehrt Tätigkeiten zu diesem Thema stattfinden und Updates folgen werden.¹⁴¹

¹³⁸ Steyer2021e

¹³⁹ Webpack2020b

¹⁴⁰ Gechev2021

¹⁴¹ Angular2021a

Das Kriterium *Nähe zum Standard* mit einer Gewichtung von 9,09% wird daher mit 2 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,1818.

Verschiedene Frameworks

Andere Frameworks sind bei Angular Module Federation nur möglich, wenn diese über Web Components eingebunden werden. Um ein Angular Modul über Module Federation einzubinden, muss das Microfrontend in Angular geschrieben sein.

Die Unterstützung für Angular ist aufgrund einiger Hilfsbibliotheken weit fortgeschritten und das Implementieren in diesem Framework hat dadurch den geringsten initialen Aufwand. Dadurch eignet sich Angular gut als Portalshell.

Das Kriterium *Verschiedene Frameworks* mit einer Gewichtung von 7,58% wird daher mit einem von fünf Punkten bewertet und erhält dadurch nur einen Teilnutzwert von 0,0758.

Entwicklungsaufland

Der initiale Entwicklungsaufwand, um Module Federation zu ermöglichen, ist höher als bei Iframes und Web Components, jedoch ebenfalls überschaubar. Um das Einrichten von Module Federation in einer Angular Applikation zu erleichtern, kann das NPM Paket `@angular-architects/module-federation` installiert werden. Dieses bietet einige Hilfestellungen zum Einbinden von anderen Module Federation Apps, ist aber nicht zwingend notwendig.

Das NPM-Paket generiert die benötigten Dateien und bereitet die Anwendung für Module Federation vor. Am wichtigsten ist dabei die `webpack.config.js`-Datei, welche sämtliche Konfigurationen für Module Federation beinhaltet. Das in Anhang 1 enthaltene Listing 4 zeigt die benötigte Konfiguration der `webpack.config.js` eines Module Federation Microfrontends, welches das eigene Modul *ExportModule* freigibt.

Das ExportModule des Microfrontends muss über eine eigene Routingkonfiguration verfügen, die im einfachsten Fall immer auf die gleiche Angular Komponente (bspw. ContentComponent) zeigt. Wird also das ExportModule aufgerufen, leitet Angular immer zur ContentComponent weiter und zeigt deren Inhalt an.

Damit der Inhalt in der Portalshell angezeigt werden kann, muss ebenfalls die `webpack.config.js` der Portalshell modifiziert werden. Dies ist in Anhang 1 im Listing 5 dargestellt. Sowohl Microfrontend, als auch Portalshell müssen die Pakete definieren,

welche geteilt werden sollen. In diesem Fall werden die Standard Angular Pakete `@angular/material`, `@angular/common`, `@angular/common/http`, `@angular/router` und `@angular/core` geteilt. Geteilte Pakete werden vom Microfrontend nicht zum Client übertragen und reduzieren so die übertragene Datenmenge.

Damit das Microfrontend über Module Federation eingebunden werden kann, muss es von der Portalshell verlinkt und aufgerufen werden, wie in Listing 6 in Anhang 1 dargestellt. Die Methode `loadRemoteModule` liest die `remoteEntry.js`-Datei an der übergebenen URL und registriert das konfigurierte Modul des Microfrontends, in diesem Fall das `ExportModule`. Anschließend kann der Router der Portalshell zum Eintrag weiterleiten und zeigt die `ContentComponent` des Microfrontends an.

Weitere Module können ebenfalls in der `webpack.config.js` zum Export freigegeben und analog auf Seiten der Portalshell mit geringem Aufwand eingebunden werden. Die Einbindung durch die Portalshell ist auch dynamisch möglich, da alle benötigten Informationen auch als Parameter übergeben werden könnten. Für eine Portalapplikation bietet es sich also an die Remoteadressen und zugehörigen Modulnamen in einer Datenbank zu speichern und diese dynamisch bei der Anzeige auszulesen.

In Tabelle 4 wurde der initiale Aufwand für Module Federation negativ bewertet. Der Anpassungsaufwand ist positiv zu bewerten, da sich Portalshell und Microfrontend Services zum Datenaustausch teilen. Und der Aufwand bei mehrfacher Einbindung ist ebenfalls positiv zu bewerten, da die Einbindungen ohne Mehraufwand skalieren.

Aufgrund des Aufwands von 2 PT werden dem Kriterium *EntwicklungsAufwand* 3 von 5 Punkten vergeben, was einen Teilnutzwert von 0,1818 ergibt.

Renderingzeit

Die Renderingzeit für Module Federation ist, wie in Tabelle 5 dargestellt, mit 39 ms sehr gering. Unter den vier verglichenen Arten der Einbindung ist Module Federation die Zweitschnellste. Nur Module Federation durch Web Components ist noch schneller.

Das Module Federation Microfrontend benötigt eine längere Ladezeit, wenn viele Bibliotheken geteilt werden.¹⁴² Dies ist darauf zurückzuführen, dass die Portalshell und

¹⁴²Siehe Abb. 59 in Anhang 2

das Microfrontend gemeinsam aushandeln müssen, welche Versionen der Bibliotheken sie verwenden und ob die Versionen kompatibel miteinander sind.

Das Kriterium *Renderingzeit* mit einer Gewichtung von 4,55% wird daher mit 4 von 5 möglichen Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,1820.

Lock-In Effekt

Mit Module Federation wird die Portalshell und das Microfrontend um ein Paket erweitert. Allerdings kann ein Module Federation Microfrontend, dessen Stärken im Routing und Teilen von großen Bibliotheken liegt, ohne nennenswerten Aufwand auch als Iframe eingebunden werden. Mit den Nachteilen, dass dann die Bibliotheken nicht mehr geteilt werden könnten und der Datenaustausch erschwert werden würde. Allerdings entschließt man sich ausschließlich das Angular Framework zu nutzen.

Es entsteht lediglich ein geringer Aufwand auf Seiten der Portalshell und des Microfrontends, wenn die installierten Module Federation Bibliotheken und Hilfskomponenten wieder entfernt werden müssen. Deswegen wird das Kriterium *Lock-In Effekt* mit einer Gewichtung von 3,03% mit 3 von 5 Punkten bewertet und erhält einen Teilnutzwert von 0,0909.

Datenmenge

Eine der Hauptfunktionalitäten von Module Federation ist das Teilen von Bibliotheken zwischen der Portalshell und dem Microfrontend.

Im vergleichbaren Test, welcher in Tabelle 5 in Abschnitt 4.4.2 dargestellt ist, hat Module Federation im einfachen Test mit 2523 KB die zweithöchste Datenmenge. Dies ist durch den zusätzlichen Code und die zusätzlich installierten Bibliotheken zu erklären.

Mit Module Federation können Bibliotheken zwischen Microfrontend und Portalshell geteilt werden. In bestimmten Konstellationen kann dadurch die übertragene Datenmenge verringert und Vorteile gegenüber den Einbindungen als Iframe und Web Component erzielt werden. Eine detaillierte Analyse der Skalierbarkeit der Datenmenge wird im nachfolgenden Abschnitt 4.4.5 bei der Evaluierung des dortigen Kriterium *Datenmenge* vorgenommen. Die Erkenntnisse sind aber auf Module Federation mit Angular Modulen gleichermaßen übertragbar.

Das Kriterium *Datenmenge* mit einer Gewichtung von 1,52% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,0608.

Zwischenfazit

Module Federation mit Angular Komponenten eignet sich, wenn die Portalshell und das Microfrontend auf Angular basieren. Durch Module Federation wird *lazy loading* von externem Code ermöglicht, welcher zur Verringerung der Kopplung sowie der Datenmenge geteilte Bibliotheken verwendet.

4.4.5 Module Federation mit Web Components

In diesem Abschnitt wird Module Federation mit Web Components als Einbindungsart eines Microfrontends in eine Portalshell zu den in Abschnitt 4.3 gewichteten Kriterien bewertet. Die Grundlagen der Einbindung gleichen dem vorherigen Abschnitt 4.4.4, allerdings wird diesmal anstatt eines Angular Modules, welches vom Microfrontend freigegeben und von der Portalshell eingebunden wird, Code als Web Component freigegeben und von der Portalshell eingebunden.

Von daher stellt dieses Kapitel eine Kombination der beiden vorherigen Evaluierungsoptionen dar und kann gegen diese verstärkt abgegrenzt sowie verglichen werden. Kriterien, welche bei dieser Art der Einbindung Module Federation vollständig gleichen, werden mit der gleichen Punktzahl versehen und zur Begründung lediglich auf die Ausführung unter Module Federation mit Angular Komponenten verwiesen.

Interoperabilität

Die Interoperabilität bei Module Federation mit Web Components ist gleichbedeutend, wie die Interoperabilität bei der standardmäßigen Einbindung durch Module Federation. Auch hier müssen Bibliotheken geschaffen werden, die den Datenaustausch zwischen Portalshell und dem Microfrontend ermöglichen.

Sollte das eingebundene Microfrontend in einem anderen Web Framework entwickelt worden sein, so muss die geteilte Bibliothek zum Datenaustausch ebenfalls eine universelle Web Component sein, weil die Lösungen ansonsten nicht kompatibel miteinander sind.

Das Kriterium *Interoperabilität* mit einer Gewichtung von 16,67% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,6668.

Wartbarkeit

Bei Module Federation sind in Zukunft noch Updates zu erwarten, weil es sich noch nicht

lange im produktiven Einsatz befindet. Web Components wiederum sind ein langjähriger Standard, bei dem in Zukunft keine größeren Updates zu erwarten sind.

Durch das Teilen von Bibliotheken untereinander ist der Wartungsaufwand leicht erhöht. Denn beim Hinzufügen von neuen Features müssen sowohl beim Microfrontend, als auch bei der Portalshell die geteilten Bibliotheken konfiguriert werden. Dadurch müssen beide eigentlich unabhängigen Instanzen neu veröffentlicht werden, wenn man vom Vorteil des Teilens von Bibliotheken profitieren möchte.¹⁴³ Dieser Aufwand ist allerdings nur geringfügig und kann für verringerte Datenmenge und verbesserte Interoperabilität in Kauf genommen werden.

Das Kriterium *Wartbarkeit* mit einer Gewichtung von 15,15% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,6060.

Unabhängige Entwicklerteams

Die Entwicklerteams sind durch Module Federation mit Web Components unabhängiger, als durch das reguläre Module Federation mit Angular Modulen. Die unabhängigen Teams können ihr Web Framework frei wählen und auch die Versionen ihrer Bibliotheken eigenständig bestimmen. Man sollte sich zwar auf der gleichen Major Version befinden, allerdings ist dies zum Teilen von Bibliotheken nicht verpflichtend. Entweder verzichtet man zum Nachteil von Datenmenge, aber zum Vorteil von Unabhängigkeit auf geteilte Bibliotheken oder man nimmt bewusst Versionsunterschiede in Kauf, welche aber durch Tests abgesichert werden können.

Daher ist das Kriterium *Unabhängige Entwicklerteams* für Module Federation durch Web Components höher zu bewerten, als im vorherigen Abschnitt 4.4.4. Es werden daher 4 von 5 Punkte vergeben, was bei der Gewichtung von 13,64% zu einem Teilnutzwert von 0,5456 führt.

Kompatibilität

Durch Module Federation mit Web Components ist eine höhere Kompatibilität zu erzielen, als bei der regulären Module Federation. Dies ist sowohl über die weiteren nutzbaren Web Frameworks zu begründen, als auch über weniger restriktive Versionsrichtlinien.

Ist die Portalshell in einer anderen Major Version des Web Frameworks gebaut, so kann es zu Problemen kommen. Die Kapselung in Web Components umgeht dieses Problem

¹⁴³Steyer2021a

und ermöglicht mehr Kompatibilität, als das Einbinden über Module Federation mit Angular Modulen.¹⁴⁴

Für Design-Komponenten, welche auf globale Styles setzen, gilt das gleiche, wie bei der Einbindung nur über Web Components. Dementsprechend wird das Kriterium *Kompatibilität* mit einer Gewichtung von 12,12% wird mit 3 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,3636.

Autonomie

Das Microfrontend, welches über Module Federation mit Web Components eingebunden wird, funktioniert auch autonom ohne die Portalshell. So kann es trotz der konfigurierten geteilten Bibliotheken eigenständig laufen, weil die Bibliotheken beim Microfrontend bereitliegen und bei Bedarf doch vom Microfrontend zum Client geladen werden. Wenn also eine fehlerhafte Konfiguration der geteilten Bibliotheken vorliegt, erkennt Webpack das Problem und exportiert die Bibliothek vom Microfrontend mit, anstatt sie von der Portalshell zu beziehen.

Das eingebundene Microfrontend kann die Portalshell nicht durch Stylings beeinflussen, wie in der Abb. 49 in Anhang 2 dargestellt ist. Das Microfrontend in dem roten Rechteck setzt die Hintergrundfarbe der Überschrift auf rot, die Portalshell, welche außerhalb des roten Rechtecks ist, setzt die Farbe auf blau. Es finden keinerlei Überschreibungen statt.

Das Kriterium *Autonomie* mit einer Gewichtung von 10,61% wird mit 4 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,4244.

Nähe zum Standard

Ebenso wie im vorherigen Abschnitt 4.4.4 ist Module Federation ebenfalls nicht der offizielle von Angular empfohlene Standard. Allerdings wird ein offiziell anerkannter Standard, nämlich Web Components, verwendet.

Das Kriterium *Nähe zum Standard* mit einer Gewichtung von 9,09% wird daher mit 3 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,2727.

Verschiedene Frameworks

Module Federation beruht auf Webpack als Modulbundler. Damit werden alle Web

¹⁴⁴Steyer2021a

Frameworks unterstützt, die auf Webpack setzen. Dazu zählen die gängigen Web Frameworks *Angular*,¹⁴⁵ *React*¹⁴⁶ und *Vue.js*¹⁴⁷. Wenn Webpack oder das dafür benötigte NodeJS nicht enthalten sind, kann Module Federation nicht eingesetzt werden. HTML-Webseiten die nicht auf einem Web Framework beruhen, sind somit nicht kompatibel.

Die Einbindung von Microfrontends, welche in einem anderen Web Frontend Framework programmiert wurden, muss über eine Web Component geschehen, da durch den öffentlichen Standard eine Basis geschaffen wird, die für jedes Web Frontend Framework gilt.¹⁴⁸

Das Kriterium *Verschiedene Frameworks* mit einer Gewichtung von 7,58% wird daher mit 3 von 5 Punkten bewertet und erhält somit einen Teilnutzwert von 0,2274.

Entwicklusaufwand

Um ein Microfrontend über Module Federation als Web Component einzubinden, sind wenige Schritte auf Seiten des Microfrontends, aber mehrere Schritte auf Seiten der Portalshell nötig.

Nach der Installation des Paketes `@angular/architects/module-federation` in beiden Applikationen, müssen ebenfalls wieder `webpack.config.js`-Dateien angepasst werden. Für die Portalshell ist die gleiche Konfiguration ausreichend, wie bei der vorherigen Einbindung durch Module Federation, welche in Anhang 1 unter Listing 5 dargestellt ist.

Die `webpack.config.js` des Microfrontends muss anstelle eines Modules die `bootstrap.ts` veröffentlichen, wie in Anhang 1 unter Listing 7 dargestellt. In der `ngDoBootstrap`-Methode der `app.module.ts` des Microfrontends wird, wie in Anhang 1 im Listing 8 dargestellt, die Angular Komponente `ContentComponent` als *CustomElement* definiert. Die `bootstrap.ts` führt in der `app.module.ts` die Kompilierung aus. Dadurch wird die Web Component generiert und kann eingebunden werden.

Zur Einbindung des Microfrontends muss ebenfalls die `loadRemoteModule`-Methode mit den Parametern des Microfrontends aufgerufen werden (siehe Listing 9 in Anhang 1).

¹⁴⁵ Steyer2020a

¹⁴⁶ ElHousieny2021

¹⁴⁷ Henry2021

¹⁴⁸ Steyer2020a

Das Aufrufen der Komponente durch die Portalshell erfolgt durch eine Hilfskomponente, welche vom Angular Router mit den Parametern `importName`, dem einzigartigen Namen des Module Federation Microfrontend, und `elementName`, dem Namen des definierten `customElements`, aufgerufen wird. Die Hilfskomponente, welche in Anhang 1 in Listing 10 dargestellt ist, ermittelt anhand des `importName` die einzubindenden Dateien und kann durch den `elementName`-Parameter die Web Component in den DOM-Tree einfügen.

In Summe ist der Aufwand von Module Federation mit Web Components geringfügig höher, als der Aufwand bei der vorherigen Module Federation Einbindung mit Angular Modulen, was auf die einmalige Erstellung der Hilfskomponenten zurückzuführen ist.

Das Kriterium *Entwicklungsauwand* mit einer Gewichtung von 6,06% wird daher mit 2 von 5 Punkten bewertet und erhält somit einen Teilnutzwert von 0,1212.

Renderingzeit

In Tabelle 5 ist die Renderingzeit für Module Federation mit Web Components mit 23 ms von allen Arten der Einbindung am niedrigsten.

In Anhang 2 in Abb. 50 ist die Ladezeit eines Microfrontends zu sehen, welches über Module Federation mit Web Components eingebunden wird und acht große Bibliotheken mit der Portalshell teilt. Die dortige Ladezeit ist mit 230ms noch ausreichend gering und wird vom Benutzer nicht negativ wahrgenommen.¹⁴⁹

Das Kriterium *Renderingzeit* mit einer Gewichtung von 4,55% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,1820.

Lock-In Effekt

Bei Module Federation mit Web Components verhält es sich gleich, wie bei der normalen Einbindung durch Module Federation. Es liegt kein Lock-In Effekt vor. Das gesamte Microfrontend kann ebenfalls als Iframe sowie auch als reine Web Component eingebunden werden.

Es entsteht ebenfalls wieder nur der geringe Aufwand des Ausbauens der Module Federation Bibliotheken.

¹⁴⁹unbounce2022

Das Kriterium *Lock-In Effekt* mit einer Gewichtung von 3,03% wird daher mit 4 von 5 Punkten bewertet und erhält dadurch einen Teilnutzwert von 0,1212.

Datenmenge

Wie in Tabelle 5 im Abschnitt 4.4.2 dargestellt, ist die gesamte Datenmenge im Standardtestfall bei Module Federation mit Web Components mit 2609 KB in Summe am höchsten. Ebenfalls wie bei der vorherigen Einbindung von Angular Modulen durch Module Federation ist die höhere Datenmenge durch die zusätzlichen Bibliotheken zu erklären. Bei Module Federation mit Web Components muss auch die gebündelte Web Component selber in Höhe von 129KB übertragen werden (siehe Abb. 33 in Anhang 2).

Ein Vorteil von Module Federation gegenüber der Einbindung als Web Component oder Iframe ist das Teilen von Bibliotheken. In der nachfolgenden Abb. 23 wurde die übertragene Datenmenge bei einer Einbindung durch Web Components mit einer Einbindung von Module Federation mit Web Components verglichen.

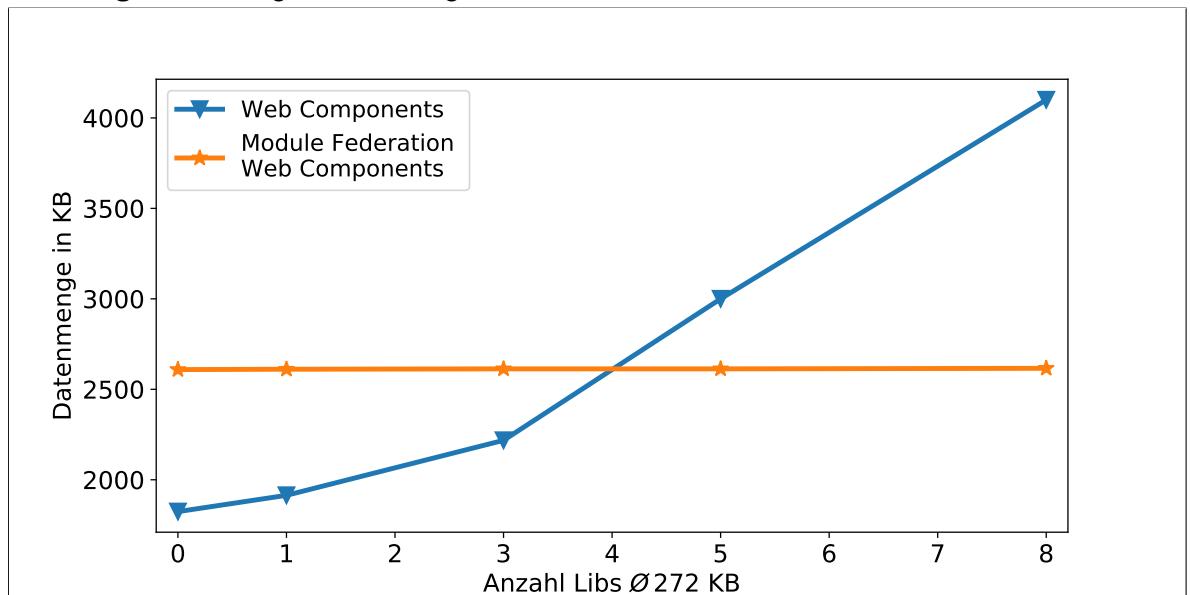
Dafür wurde zunächst die durchschnittliche Paketgröße der 50 populärsten NPM Bibliotheken ermittelt.¹⁵⁰ Die durchschnittliche Paketgröße beträgt 262,9KB. Der Median der 50 Bibliotheken liegt bei 63KB. Die Standardabweichung liegt bei 615, was auch im Histogramm in Abb. 55 in Anhang 2 zu sehen ist. Der stark abweichende Median von der durchschnittlichen Größe ist durch eine geringe Anzahl von sehr großen Bibliotheken im niedrigen Megabyte (MB) Bereich zu erklären. Die Mehrheit der NPM Bibliotheken ist kleiner als 100 KB. Aber eben genau die ausschlaggebenden großen Bibliotheken bieten sich gut zum Teilen an, damit sie nicht doppelt geladen und übertragen werden müssen.

In Tabelle 7 in Anhang 3 ist eine Auflistung von 8 exemplarischen NPM Bibliotheken zu sehen, welche zum Zeitpunkt der Bearbeitung über eine annähernde durchschnittliche Datenmenge, wie der Gesamtdurchschnitt verfügen. Der Durchschnitt der 8 beispielhaften Bibliotheken liegt bei 272.8KB. In der nachfolgenden Abb. 23 wurden die ausgewählten Bibliotheken in einem Microfrontend als Web Component und in einem Microfrontend durch Module Federation mit Web Components in eine Portalshell eingebunden. Die Messungen wurden ebenfalls unter den in Anhang 2 in Abb. 57 dargestellten Laborbedingungen durchgeführt. Die einzelnen Nachweise der Messergebnisse

¹⁵⁰Kashcha2019

sind in Anhang 5 unter dem fünften Punkt *Messergebnisse geteilte Bibliotheken Web Components* zu finden.

Abbildung 23: Übertragene Datenmenge bei mehreren Bibliotheken



Quelle: Eigene Darstellung

Es wurden 5 Messungen mit null, einer, drei, fünf und acht verwendeten Bibliotheken durchgeführt. In Abb. 38 in Anhang 2 sind die Bibliotheken pro Messlauf mit der durchschnittlichen Paketgröße dargestellt. Die vorherige Abb. 23 zeigt, dass ab 800KB, welche durch Bibliotheken übertragen werden, Module Federation die bessere Art der Einbindung ist, um übertragene Datenmengen einzusparen. Die blaue Linie der Web Components Datenmenge weist einen leichten Knick am dritten Messpunkt auf, weil die eingebundenen Bibliotheken nicht alle genau dem Durchschnitt entsprechen.¹⁵¹ Wären sie alle gleich groß, so würde der Graph linear verlaufen.

Allerdings ist das Teilen von Bibliotheken nur von Vorteil, wenn die Bibliothek ohnehin schon von der Portalshell geladen werden musste. Dies ist beispielsweise bei Standardbibliotheken wie `@angular/core` oder einer geteilten Fachbibliothek, welche sowohl Microfrontend als auch die Portalshell benötigen, der Fall. Ebenfalls ist es auch vorstellbar, dass mehrere Microfrontends die gleiche Bibliothek benötigen, die Portalshell aber ursprünglich nicht. Dann könnte die Portalshell so umgebaut werden, dass die mehrfach benötigten Bibliotheken einmalig von der Portalshell eingebunden werden

¹⁵¹Vgl. Abb. 38 in Anhang 2

und sämtliche Microfrontends dann auf die eine eingebundene Bibliothek zugreifen, um Datenmenge zu sparen.

Module Federation kann gegenüber Web Components oder einem Iframe Vorteile bei der übertragenen Datenmenge bringen, jedoch nur im besonderen Anwendungsfall. Bei kleinen Microfrontends ohne abhängigen Bibliotheken sind Iframes und Web Components im Vorteil. Bei großen Lösungen mit vielen Bibliotheken, lohnt sich die Einbindung über Module Federation wiederum.

Das Kriterium *Datenmenge* mit einer Gewichtung von 1,52% wird daher mit 4 von 5 Punkten bewertet und erhält somit einen Teilnutzwert von 0,0608.

Zwischenfazit

Module Federation mit Web Components überträgt die Vorteile der Module Federation Einbindung von Angular Modulen auf andere Web Frameworks. Dadurch ist die Kompatibilität gegenüber anderen Frameworks hoch. Ebenfalls positiv anzumerken sind geringe Renderingzeiten sowie die Reduzierung der übertragenen Datenmenge, besonders bei geeigneten Konstellationen. Module Federation mit Web Components hat von den vier verglichenen Arten der Einbindung den höchsten Einrichtungsaufwand und ist ebenfalls kein offiziell anerkannter Standard von Angular.

4.5 Vergleich der Arten untereinander

In diesem Abschnitt werden die im vorherigen Kapitel Abschnitt 4.4 festgestellten Ausprägungen für jedes Kriterium jeder Alternative miteinander verglichen, um somit die beste Lösung für jedes Kriterium festzustellen.

In der nachfolgenden Abb. 24 sind die einzelnen Teilnutzwerte je Art der Einbindung und Kriterium in einer Tabelle gegenübergestellt. Die Teilnutzwerte sind gruppiert pro Kriterium in einer Farbskala dargestellt, damit der höchste Teilnutzwert (in grün) je Kriterium erkennbar ist.

Abbildung 24: Gegenüberstellung Ergebnisse der Nutzwertanalyse

Kriterien	Alternativen			
	IFrame	Web Components	Module Federation	Module Fed. mit Web Components
Interoperabilität	0,1667	0,6668	0,8335	0,6668
Wartbarkeit	0,4545	0,6060	0,6060	0,6060
Unabh. Entwicklerteams	0,5456	0,4092	0,4092	0,5456
Kompatibilität	0,1212	0,3636	0,3636	0,3636
Autonomie	0,5305	0,4244	0,4244	0,4244
Standard	0,4545	0,4545	0,1818	0,2727
Versch. Frameworks	0,3790	0,3032	0,0758	0,2274
Entwicklungsaufwand	0,2424	0,2424	0,1818	0,1212
Renderingzeit	0,0910	0,1365	0,1820	0,1820
Lock-In Effekt	0,1515	0,1515	0,0909	0,1212
Datenmenge	0,0152	0,0608	0,0608	0,0608
Nutzwerte	3,1521	3,8189	3,4098	3,5917

Quelle: Eigene Darstellung

Für den gegebenen Anwendungsfall hat die Web Component bei der Evaluierung den höchsten Nutzwert erreicht. Dennoch konnten Web Components nicht bei allen Kriterien den höchsten Teilnutzwert erzielen, denn die direkte Gegenüberstellung der Einbindungsarten zeigt, dass keine der vier Arten für alle elf Kriterien am besten geeignet ist.

Dies lässt darauf schließen, dass die Arten der Einbindung abhängig nach ihrem gegebenen Anwendungsfall optimal sind. Eine Übersicht, welche Art der Einbindung bei welchem Anwendungsfall optimal geeignet ist, wird im nachfolgenden Abschnitt 4.6 dargestellt.

4.6 Optimale Anwendungsszenarien je nach Art der Einbindung

In diesem Abschnitt werden Anwendungsszenarien mit den dazugehörigen Problemstellungen den Kriterien zugeteilt, danach mit den Arten der Einbindung verglichen und auf Basis der Ergebnisse ein Entscheidungsleitfaden gebildet.

Die nachfolgende Tabelle 6 zeigt vier Anwendungsfälle, welche verschiedene Problemstellungen aufweisen und dadurch unterschiedlich viel Wert auf einzelne Kriterien legen.

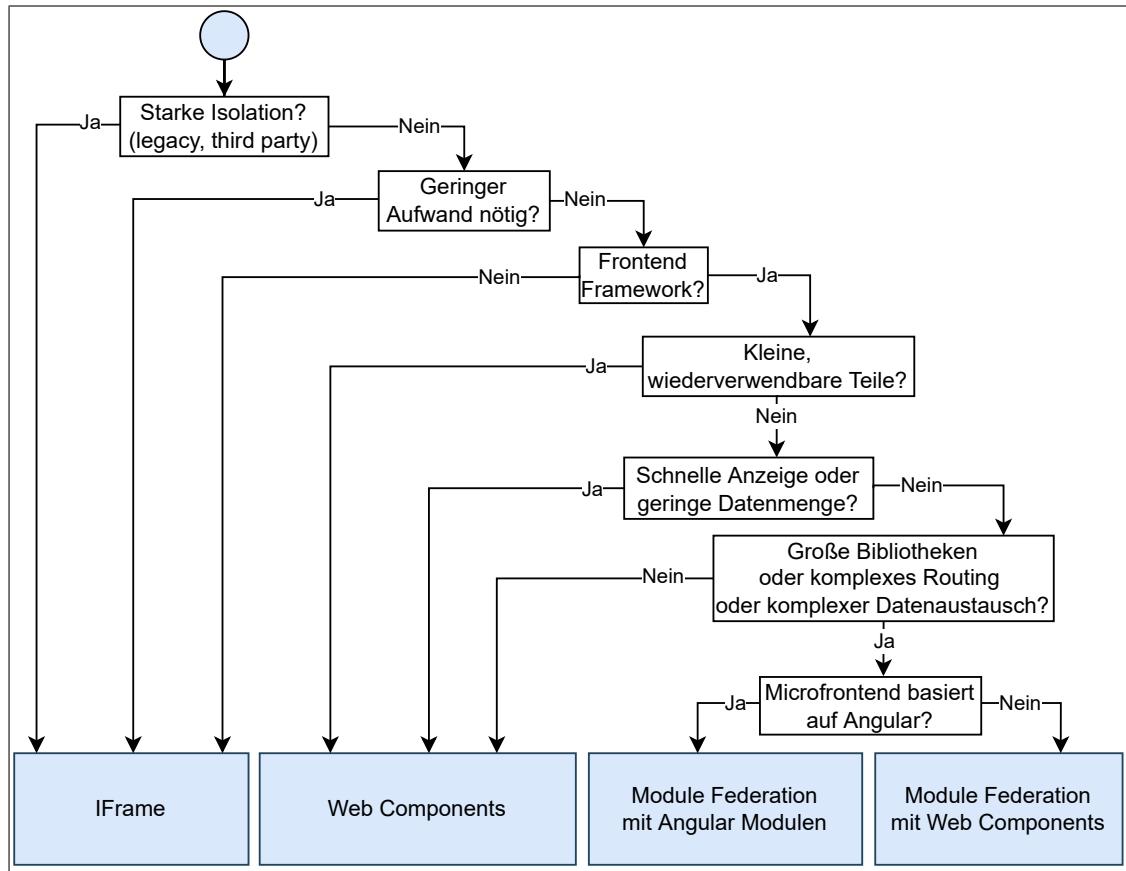
Tabelle 6: Zusammenhang Kriterium zu Anwendungsfall

Anwendungsfall	Beispielhafte Problemstellung	Kriterien
Legacy Applikation	Veraltete Frameworks, wenig KnowHow, keine Investition, Kundenanforderung	Entwicklungsaufwand, Kompatibilität
Moderne Applikation, geringer Umfang	ReactJS als Framework, geringe Datenmenge nötig, Komplexität	Datenmenge, Frameworks
Moderne Applikation, mittlerer Umfang	Wiederverwendbar, Neuentwicklung, universell, Interoperabilität	Entwicklungsaufwand, Frameworks
Moderne Applikation, großer Umfang	Angular als Framework, viel Routing, große Bibliotheken	Interoperabilität, Kompatibilität, Datenmenge

Quelle: Eigene Darstellung

Für vier Anwendungsfälle muss die jeweils beste Lösung zur Einbindung in die Portalshell gefunden werden. Dafür wird ein Entscheidungsbaum erstellt, welcher sich an dem von Geers im Grundlagenteil der Thesis in Abb. 14 orientiert, aber auf den Prototyp angepasst und durch die Erkenntnisse aus Abb. 24 optimiert ist. Der entwickelte Entscheidungsbaum ist in der nachfolgenden Abb. 25 für Microfrontends zur Einbindung in eine Angular Portalapplikation dargestellt.

Abbildung 25: Entscheidungsbaum: Einbindung Microfrontend in Portalshell



Quelle: Eigene Darstellung

Die Nutzwertanalyse hat ergeben, dass Iframes sich gut eignen, wenn Microfrontends eingebunden werden sollen, die Altlasten sind, nicht viel Aufwand erzeugen sollen, kein Web Frontend Framework enthalten oder stark gegen andere Microfrontends isoliert werden müssen.

Ist der Use-Case des Microfrontends klein und wiederverwendbar, bspw. die konfigurierbaren Anzeige eines KPI, oder kommt es auf schnelle Anzeige oder geringe übertragene Datenmenge bei wenig Bibliotheken an, so sind Web Components zu empfehlen.

Bei großer Datenmenge durch Bibliotheken, bei komplexem Routing im Microfrontend oder intensivem Datenaustausch mit der Portalshell empfiehlt sich die Einbindung über Module Federation. Abhängig davon, ob das Web Framework des Microfrontends Angular ist, sollte Module Federation mit Web Components oder mit Angular Modulen gewählt werden.

5 Prototypische Implementierung

In diesem Kapitel werden die Erkenntnisse aus dem vorherigen Kapitel 4 auf das prototypische Beispiel aus Kapitel 3 angewendet. Dafür wird zunächst der Anpassungsbedarf an der Beispielapplikation erläutert, dann die Microfrontends eingebunden und anschließend zur Kontrolle des erstellten Prozesses die Ergebnisse validiert.

Die drei Microfrontends aus Kapitel 3 (Wetter-App, Taschenrechner-App und Statistik-App) werden anhand des in Abb. 25 dargestellten Entscheidungsbaumes der jeweiligen passenden Art der Einbindung zugewiesen.

Die Wetter-App soll für jeden Mandanten wiederverwendet und in einem Dashboard mehrfach eingebunden werden können, damit das Wetter unterschiedlicher Städte angezeigt werden kann. Es handelt sich um eine Eigenentwicklung in Angular ohne große Komplexität und eigenes Routing. Daher wird sie über Web Components eingebunden.

Ein Taschenrechner ist online frei verfügbar abrufbar unter <https://www.blitzrechner.de/taschenrechner-wissenschaftlich/>. Dieser Rechner bildet sämtliche Funktionalitäten ab, sodass der Aufwand für eine Eigenentwicklung nicht gerechtfertigt wäre. Von daher wird der Taschenrechner des Drittanbieters als Iframe eingebunden.

Die Statistik-App ist eine Eigenentwicklung in Angular. Sie beinhaltet große Bibliotheken zum Darstellen von Graphen und Berechnen von Statistiken. Deswegen ist eine Einbindung über Module Federation mit Angular Modulen zu empfehlen. Die App teilt sich Bibliotheken mit der Portalshell und bekommt durch eine **AuthLib** Informationen über den User von der Portalapplikation, damit das Rechtekonzept umgesetzt werden kann.

5.1 Anpassungsbedarf der Beispielapplikation

Bei der prototypischen Beispielapplikation aus Kapitel 3 liegt Anpassungsbedarf nach den Erkenntnissen aus der vorangegangenen Evaluierung vor. Die Portalshell muss erweitert werden, sodass Microfrontends als Iframe, Web Component und Module Federation mit Angular Modulen eingebunden werden können. Die Portalshell sollte dafür eine Oberfläche bereitstellen, welche das Einbinden der Microfrontends dynamisch ermöglicht

und die Konfiguration in einer Datenbank speichert. Auf das dynamische Einbinden und das Entwickeln einer Oberfläche zum Administrieren wird verzichtet, da dies den Rahmen eines Prototypen übersteigen würde.

Allerdings wird die Portalshell um drei Komponenten erweitert, die das Einbinden von Microfrontends anhand der nötigen Parameter ermöglichen. So wird gezeigt, dass das Einbinden dynamisch geschehen könnte, wenn die Komponenten dynamisch vom Backend gefüllt werden würden.

Das Designmockup aus Abb. 18 in Anhang 2 sieht eine horizontale Trennung der Microfrontend-Architektur vor.¹⁵² So sind mehrere Microfrontends, die von verschiedenen Teams verwaltet werden, auf einem Dashboard eingebunden. Der Inhalt des angezeigten Dashboards besteht dann aus vier Microfrontendinstanzen von drei unterschiedlichen Microfrontend-Templates. Die Komponenten zum Einbinden der Microfrontendinstanzen geben das Styling zur Ausrichtung auf dem Dashboard vor.

5.2 Implementierung der Microfrontends

Das Taschenrechner-Microfrontend konnte mit geringem Aufwand als Iframe eingebunden werden. Dadurch, dass der Taschenrechner bereits abrufbar zur Verfügung stand, wurde Entwicklungsaufwand vermieden. Für die Einbindung war es ausreichend die URL des Taschenrechner-Microfrontends mit Breite- und Höhe-Parametern an die *Iframe-Injection-Komponente*¹⁵³ der Portalshell zu übergeben. Das durch die *Iframe-Injection-Komponente* generierte HTML ist in Anhang 1 in Listing 13 zu sehen.

Die Wetter-App wurde als Angular Komponente entwickelt und dann als Web Component exportiert. Statt wirklich eine Wetter-API anzusprechen und die Wetterdaten des Standort-Parameters abzufragen, wird lediglich der Standort prototypisch angezeigt. Der Sourcecode der *Weather-Component* ist in Anhang 1 in Listing 16 dargestellt. Die Web Component wurde automatisch gebaut und die generierte `main.js`-Datei in einem Azure Storage Account abgelegt.

¹⁵²Siehe Abb. 7

¹⁵³Sourcecode in Anhang 1 in Listing 14

Die Portalshell nimmt die URL zur `main.js` entgegen und kann durch die *WC-Injection-Component*¹⁵⁴ die *Weather-Component* in den DOM-Tree des Dashboards einbinden.

Die Statistik-App wurde selber entwickelt und über Module Federation in die Portalshell eingebunden. Dafür wurde die Routing-Konfiguration so angepasst, dass das Module Federation Microfrontend in ein zweites Router-Outlet mit dem Namen `modulefederation` gerendert wird, welches in der Seite eingebunden ist (siehe Listing 17 in Anhang 1). Für jedes Module Federation Microfrontend, das über ein Angular Modul eingebunden wird, muss eine Route hinterlegt werden. Das Hinterlegen kann auch dynamisch mit übergebenen Variablen passieren.

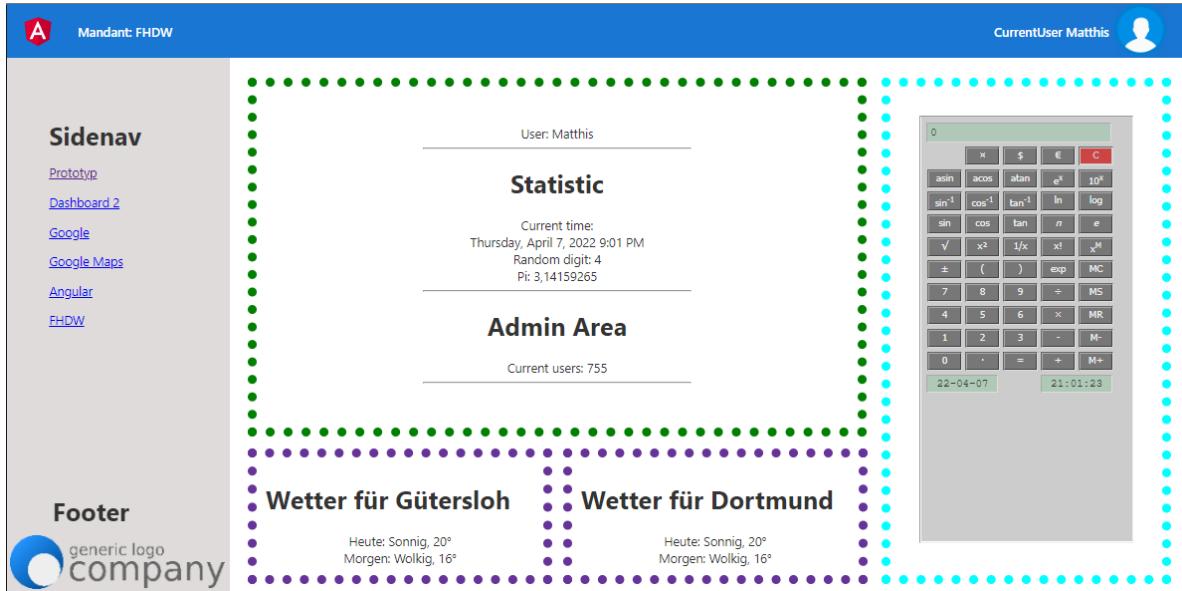
Das Einbinden einer Web Component über Module Federation könnte mit einer `Module-Federation-WC-Injection-Component` geschehen, die den Code der geladenen Web Component in den DOM-Tree des aktuellen Dashboards automatisch einfügt. Aber da die Portalshell und das Microfrontend beide in der aktuell neusten Angular Version entwickelt wurden, ist dies nicht notwendig.

5.3 Validierung der Kriterien

In der nachfolgenden Abb. 26 ist die finale prototypische Implementierung der in Kapitel 3 beschriebenen Beispiellapplikation zu sehen. Da es sich um eine Implementierung mit Fokus auf der technischen Machbarkeit handelt, wurde das Design vernachlässigt. Dies könnte mit einem Bootstrap Theme¹⁵⁵ einheitlich nachgezogen werden.

¹⁵⁴Sourcecode in Anhang 1 in Listing 15

¹⁵⁵Siehe beispielsweise <https://themes.getbootstrap.com/>

Abbildung 26: Prototypische Implementierung des Dashboard *Prototyp*

Quelle: Eigene Darstellung

Die implementierte Lösung gleicht dem Mockup aus Abb. 18, welche in Kapitel 3 vor gestellt wurde. Die vier Microfrontendinstanzen sind nebeneinander dargestellt. Eine horizontale Trennung wurde ermöglicht, sodass mehrere Instanzen auf einer Seite darge stellt werden können. Die Microfrontends sind zur besseren Darstellung der Trennung mit einem farblichen Rahmen dargestellt.

Das Module Federation Microfrontend (grün) teilt einige seiner Bibliotheken mit der Portalshell. Zum jetzigen Zeitpunkt wird dadurch kaum Datenmenge eingespart. Aber wenn weitere Statistik-Apps eingebunden werden würden, würden diese von der Teilung der Bibliotheken profitieren. Die Statistik-App kann ihr Rechtekonzept anwenden. Sie bezieht die Informationen des angemeldeten Benutzers aus einer mit der Portalshell geteilten AuthLib¹⁵⁶. Die AuthLib wird beim Login mit den Userdaten gefüllt und Module Federation Microfrontends können sie einbinden und Daten konsumie ren.

Die geteilten Userinformationen (Benutzername: *Matthis*) sind sowohl im Microfrontend, als auch im Header der Portalshell dargestellt und werden beide Male dynamisch aus der AuthLib bezogen. Da der User *Matthis* die Rolle Admin zugeordnet hat, sieht er den

¹⁵⁶Steyer2021f

erweiterten Admin-Bereich in der Statistik-App. Andere Komponenten können ebenfalls Daten aus der **AuthLib** konsumieren.

Die Wetter-App (lila) wurde doppelt mit unterschiedlichen Standorten als Web Component eingebunden. So konnte die Komponente nach einmaligem Einbinden wiederverwendet werden, ohne ein zweites Mal zum Client geladen zu werden.¹⁵⁷

Das Taschenrechner Microfrontend (blau) konnte von <https://www.blitzrechner.de/taschenrechner-wissenschaftlich/> als Iframe eingebunden werden. Der benötigte Aufwand zum Einbinden der verschiedenen Arten skalierte mit den geschätzten Erkenntnissen des Aufwands von Tabelle 5 aus Abschnitt 4.4.2 und war dementsprechend gering.

Fazit

Der in Kapitel 4 durch die Nutzwertanalyse entwickelte Entscheidungsbaum konnte an den Microfrontends des prototypischen Beispiels durchlaufen werden. Die Microfrontends wurden dadurch jeweils nach der geeignetsten Art in die Portalshell eingebunden. Durch die Erkenntnisse aus Abschnitt 5.3 konnte bestätigt werden, dass der entwickelte Entscheidungsbaum anwendbar ist und sinnvolle Resultate erzielt.

¹⁵⁷Siehe Abb. 43 in Anhang 2

6 Schlussbetrachtung

In diesem Kapitel wird eine Schlussbetrachtung zu der erstellten Masterthesis durchgeführt. Dafür wird im nachfolgenden Abschnitt ein Fazit über die Erkenntnisse erstellt und im darauffolgenden Abschnitt 6.2 potentielle weitere Themen erläutert, welche noch weiterführend betrachtet werden könnten.

6.1 Zusammenfassung

Ziel der Arbeit war es, eine differenzierte Entscheidungshilfe zu erstellen, anhand derer die optimale Art der Einbindung von Microfrontends in eine Portalshell ermittelt werden kann.

Es wurde die Erkenntnis gesammelt, dass Web Components den größten Nutzwert im Vergleich der Einbindungsarten untereinander erreichen. Allerdings schneiden Web Components in einigen Kriterien schlechter ab, als andere Arten der Einbindung. Jede der vier Einbindungsarten konnte bei mindestens fünf Kriterien den höchsten Teilnutzwert erreichen. Dies zeigt, dass eine Einzelfallbetrachtung basierend auf den individuellen Anforderungen jedes Microfrontends durchgeführt werden muss, um die optimale Art der Einbindung in eine Portalshell bestimmen zu können. Dafür wurde eine Entscheidungshilfe in Form eines Entscheidungsbaumes im letzten Abschnitt 4.6 der Nutzwertanalyse erstellt.

Durch den in Kapitel 3 beschriebenen Prototypen wurden die Erkenntnisse der Nutzwertanalyse auf die zutreffenden individuellen Ansprüche des Prototypen und dessen Microfrontends angepasst. Im vorletzten Kapitel 5 *Prototypische Implementierung* wurde das Ergebnis der Nutzwertanalyse und der daraus entstandene Entscheidungsbaum angewendet und überprüft. Es stellte sich heraus, dass die prototypische Portalshell von den Erkenntnissen profitieren konnte und die verschiedenen Arten der Einbindung ihre Vorteile an den Microfrontends anwenden konnten.

6.2 Ausblick

Die Frontend-Entwicklung sowie die Softwarearchitektur sind umfangreiche Themengebiete. Es gibt noch weitere Optionen, welche tiefer untersucht und eingebracht werden

könnten.

Zu diesen Themen zählt beispielsweise die Ermittlung von optimalen Hostingszenarien für Microfrontends. Es könnte On-Premise Hosting mit dem Hosting bei Cloud Service Providern verglichen werden. Ein weiterer möglicher Blickwinkel wäre der Vergleich verschiedener Cloud Hosting Optionen untereinander. So könnten im Kubernetes-Cluster gehostete Microfrontends gegenüber Microfrontends gehostet in einer Virtuellen Maschine (VM) gegenüber Hosting in einer *Platform as a Service (PaaS)*-Lösung eines Cloud-Anbieters zu diversen Kriterien (bspw. Kosten, Ausfallsicherheit, Unabhängigkeit und Skalierbarkeit) verglichen werden.

Ebenfalls könnte das Konzept eines Design Systems näher untersucht werden. Ein einheitliches Design in verteilten, autonomen Teams durchzusetzen, ist nicht trivial, kann aber viele Vorteile für die Entwickler sowie die Benutzer der Applikation bringen. Michael Geers gibt im zwölften Kapitel seines Buches *Micro Frontends in Action* Denkanstöße und Sichtweisen, welche Potential für eine wissenschaftliche Aufarbeitung bieten.¹⁵⁸

Das Konzept einer Portalshell wurde im Rahmen der Masterthesis nur prototypisch beschrieben und umgesetzt. Eine mandantenfähige Portalshell basierend auf einer dynamischen Microfrontend-Architektur hat das Potential die Softwarelösungen für eine ganze Branche abzubilden. Dafür müssten allerdings aufwändige Prozesse entwickelt und umgesetzt werden. Dies würde von dynamischer Instanziierung der Microfrontends je Mandant mit automatisiertem Deployment bis hin zu der Erstellung eines zentralen Monitoring- sowie Abrechnungskonzeptes reichen.

¹⁵⁸Geers2020

Anhang

Anhangsverzeichnis

Anhang 1:	Listings	103
Anhang 2:	Bilder	118
Anhang 3:	Tabellen	134
Anhang 4:	Expertengespräche	135
Anhang 5:	Sonstige Dokumente	137

Anhang 1 Listings

Listing 2: Exemplarische Konfiguration Module Federation Portalshell

```
[...]
plugins: [
  new ModuleFederationPlugin({
    remotes: {
      "microapp": "microapp@http://localhost:3000/remoteEntry.js",
    },
    shared: share({
      "@angular/core": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common/http": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/router": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      ...sharedMappings.getDescriptors()
    })
  },
  sharedMappings.getPlugin()
],
[...]
```

Auszug aus der `webpack.config.js` einer Portalshell

Listing 3: Exemplarische Konfiguration Module Federation Microfrontend

```
[...]
plugins: [
  new ModuleFederationPlugin({
    name: "microapp",
    filename: "remoteEntry.js",
    exposes: {
      './Module': './src/app/distant/distant.module.ts',
    },
    shared: share({
      "@angular/core": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common/http": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/router": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },

      ...sharedMappings.getDescriptors()
    })
  }),
  sharedMappings.getPlugin()
],
[...]
```

Auszug aus der `webpack.config.js` eines Microfrontends

Listing 4: Konfiguration eines Module Federation Microfrontends

```
[...]
plugins: [
  new ModuleFederationPlugin({
    library: { type: "module" },

    name: "evalmf",
    filename: "remoteEntry.js",
    exposes: {
      './Module': './src/app/export/export.module.ts',
    },
    shared: share({
      "@angular/core": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common/http": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/router": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/material": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      ...sharedMappings.getDescriptors()
    })
  }),

  sharedMappings.getPlugin()
],
[...]
```

Auszug aus der webpack.config.js eines Microfrontends

Listing 5: Konfiguration einer Module Federation Portalshell

```
[...]
plugins: [
  new ModuleFederationPlugin({
    library: { type: "module" },
    shared: share({
      "@angular/core": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common/http": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/router": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/material": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      ...sharedMappings.getDescriptors()
    })
  }),
  sharedMappings.getPlugin()
],
[...]
```

Auszug aus der `webpack.config.js` der Portalshell

Listing 6: Einbindung eines Module Federation Microfrontend

```
// decl.d.ts
declare module 'evalmf/Module';

// app.routes.ts
const URL = 'http://localhost:4202/remoteEntry.js';
{
  path: 'EvalMF',
  loadChildren: () => loadRemoteModule({
    type: 'module',
    remoteEntry: URL,
    exposedModule: './Module'
  })
  .then(m => m.ExportModule)
},
```

Auszüge aus der `decl.d.ts` und `app.routes.ts` der Portalshell

Listing 7: Konfiguration von Module Federation mit Web Components

```
[...]
plugins: [
  new ModuleFederationPlugin({
    name: "evalmfwc",
    library: { type: "var", name: "evalmfwc" },
    filename: "remoteEntry.js",
    exposes: {
      './web-components': './src/bootstrap.ts',
    },
    shared: share({
      "@angular/core": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/common/http": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/router": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      "@angular/material": { singleton: true, strictVersion: true,
        ↪ requiredVersion: 'auto' },
      ...sharedMappings.getDescriptors()
    })
  })
],
[...]
```

Auszug aus der `webpack.config.js` eines Microfrontends

Listing 8: Definition des customElements zur Einbindung als Web Component

```
[...]
ngDoBootstrap() {
    const webComponent = createCustomElement(ContentComponent, {injector:
        ↪ this.injector});
    //avoid double registration errors
    if(!customElements.get('content-component-mf')){
        customElements.define('content-component-mf', webComponent);
    }
}
[...]
```

Auszug aus der `app.module.ts` des Microfrontends

Listing 9: Laden einer Web Component durch Module Federation

```
// decl.d.ts
declare module 'evalmfwc/web-components';

// registry.ts
export const registry = {
  evalmfwc: () => loadRemoteModule({
    type: 'script',
    remoteEntry: 'http://localhost:4204/remoteEntry.js',
    remoteName: 'evalmfwc',
    exposedModule: './web-components'
  })
};

// app.routes.ts - Portalshell
[...]
{
  path: 'EvalMFWC',
  component: WrapperComponent,
  pathMatch: 'full',
  data: { importName: 'evalwcmf', elementName: 'content-component-mf' }
},
[...]
```

Inhalte der `decl.d.ts`, `registry.ts` und Ausschnitte der `app.routes.ts` einer Portalshell

Listing 10: Einbindung einer Web Component durch Module Federation

```

@Component({
  template: '<div #vc></div>',
})
export class WrapperComponent implements AfterContentInit {

  @ViewChild('vc', {read: ElementRef, static: true})
  vc: ElementRef;

  constructor(private route: ActivatedRoute) { }

  ngAfterContentInit(): void {
    const elementName = this.route.snapshot.data.elementName;
    const importName = this.route.snapshot.data.importName;

    const importFn = registry[importName];
    importFn()
      .then(_ => console.debug(`element ${elementName} loaded!`))
      .catch(err => console.error(`error loading ${elementName}:`, err));

    const element = document.createElement(elementName);
    this.vc.nativeElement.appendChild(element);
  }
}

```

Inhalt der `wrapper.component.ts` der Portalshell

Listing 11: Platzhalter für SSI

```
<!--#include virtual="/url/to/include" -->
```

Quelle: Geers2020

Listing 12: Nutzung eines HTML Templates

```
<button onclick="showContent()">Show hidden content</button>

<template>
<h2>Flower</h2>

</template>

<script>
function showContent() {
    var temp = document.getElementsByTagName("template")[0];
    var clon = temp.content.cloneNode(true);
    document.body.appendChild(clon);
}
</script>
```

Quelle: W3Schools2022

Listing 13: Einbindung eines Taschenrechner als Iframe

```
[...]
<iframe id="iframeId" src="https://www.blitzrechner.de/calculations/
    ↪ zf2Calculator/zf20.htm" name="Taschenrechner"
    width="220" height="310" style="border: none"></iframe>
[...]
```

Auszug aus dem generierten HTML des Dashboard *Prototyp*

Listing 14: Quellcode der IframeInjection-Component

```
import { Component, Input, OnInit } from '@angular/core';
import * as $ from "jquery";

@Component({
  selector: 'iframe-injection',
  template: '<iframe id="iframeId" [height]="height" [width]="width"></
    ↪ iframe>',
})
export class IFrameInjectionComponent implements OnInit {

  @Input()
  url: string = '';
  @Input()
  height: string = '';
  @Input()
  width: string = '';

  ngOnInit(): void {
    $('#iframeId').attr('src', this.url);
  }
}
```

Inhalt der `Iframe-injection.component.ts` der Portalshell

Listing 15: Komponente zur Einbindung von Web Components

```

import { Component, ElementRef, Input, OnInit } from '@angular/core';

@Component({
  selector: 'web-component-injector',
  template: '',
})
export class WebComponentInjectorComponent implements OnInit {
  private element: HTMLElement;

  @Input() url: string = '';
  @Input() name: string = '';
  @Input() location: string = '';

  constructor(private wrapperElement: ElementRef) { }

  ngOnInit(): void {
    this.element = this.loadWCFromUrl(this.wrapperElement, this.url,
      ↪ this.name );
  }

  loadWCFromUrl(wrapper: ElementRef, hostingURL: string, htmlTagName:
    ↪ string) : HTMLElement {
    const wrapperElement: HTMLElement = wrapper.nativeElement;
    const shadow = wrapperElement.attachShadow({mode: 'open'});
    const script = document.createElement('script');
    script.src = hostingURL;

    const webComponent = document.createElement(htmlTagName);
    webComponent.setAttribute('location', this.location);

    script.onload = () => { shadow.appendChild(webComponent); }

    shadow.appendChild(script);
    return webComponent;
  }
}

```

Inhalt der `web-component-injector.component.ts` der Portalshell

Listing 16: Quellcode der Weather-Component

```
import { Component, Input, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-weather',
  template: `
    <div class="wrapper">
      <h1>Wetter fuer {{location}}</h1>
      <div class="center">
        <div>
          <label>Heute: Sonnig, 20 Grad</label>
        </div>
        <div>
          <label>Morgen: Wolzig, 16 Grad</label>
        </div>
      </div>
    </div>`,
  styles: ['.wrapper{width: 300px;margin: auto;}', '.center{display:
    inline;float: none;text-align: center;}'],
  encapsulation: ViewEncapsulation.ShadowDom
})
export class WeatherComponent {

  @Input()
  location: string = '';

}
```

Inhalt der `weather.component.ts` des Wetter-Microfrontends

Listing 17: Einbindung des Module Federation Microfrontends

```
// app.routes.ts
[...]
{
  path: 'Prototyp',
  component: PrototypComponent,
  pathMatch: 'full',
  children: [
    {
      path: '',
      outlet: 'modulefederation',
      loadChildren: () => loadRemoteModule({
        type: 'module',
        remoteEntry: URL,
        exposedModule: './Module'
      })
      .then(m => m.ExportModule)
    }
  ],
  [...]
}

// prototyp.component.html
[...]
<div class="linksoben">
  <router-outlet name="modulefederation"></router-outlet>
</div>
[...]
```

Auszüge der `app.routes.ts` und `prototyp.component.html` der Portalshell

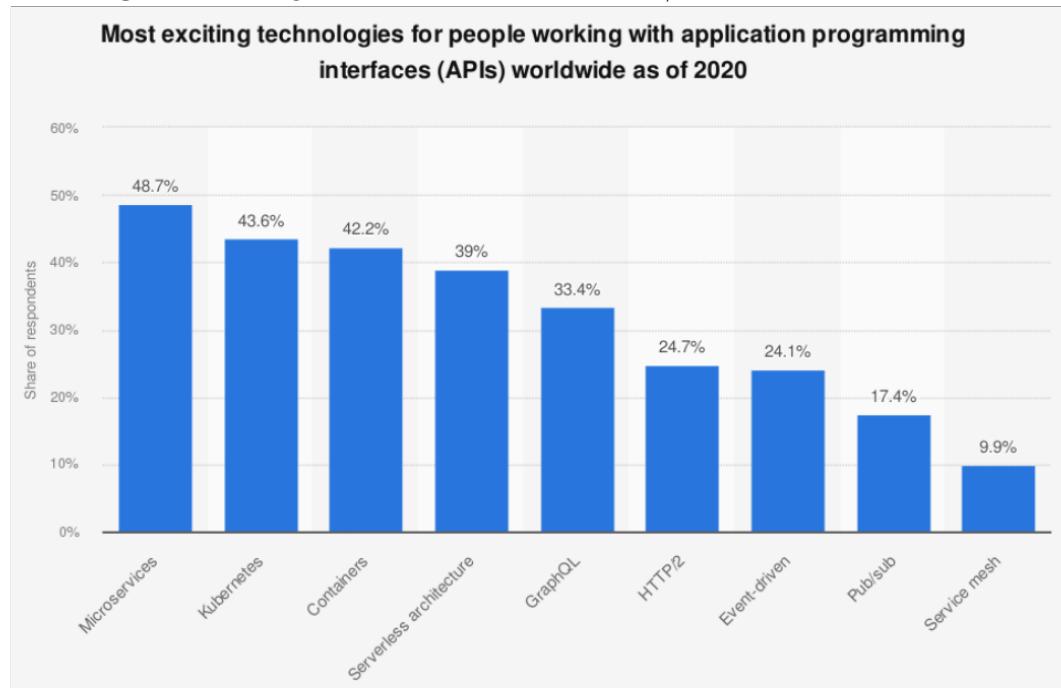
Listing 18: Einbindung einer Content-Component als Web Component

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>WebComponent</title>
  <base href="/">
  [...]
</head>
<body class="mat-typography">
  <content-component></content-component>
</body>
</html>
```

Auszüge der `index.html` des Web Component Microfrontends

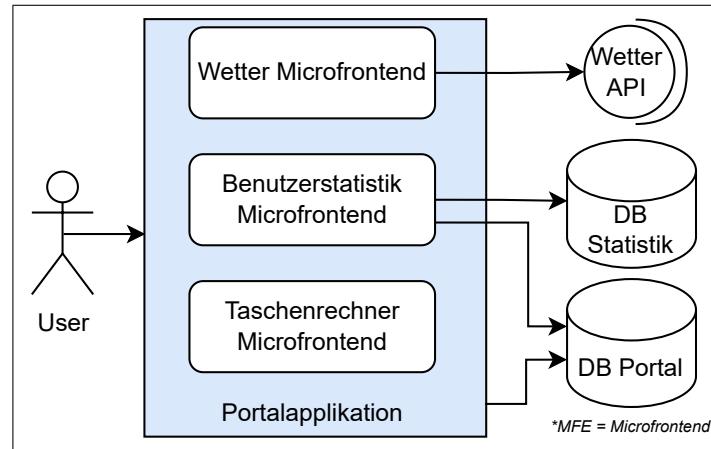
Anhang 2 Bilder

Abbildung 27: Technologien für API Entwickler, Stand 07/2020

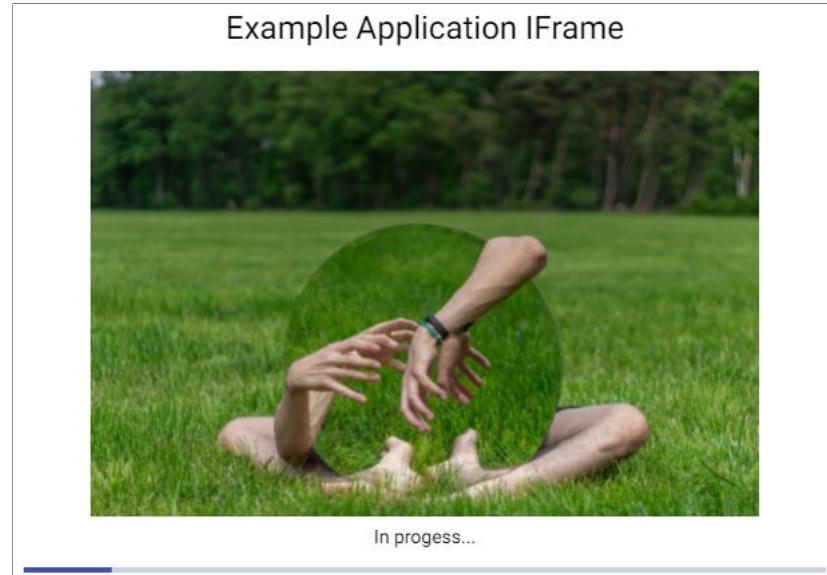


Quelle: Statista2020a

Abbildung 28: Konkrete Solutionarchitektur Beispielapplikation



Quelle: Eigene Darstellung

Abbildung 29: Vergleichbares Microfrontend für Evaluierung

Quelle: Eigene Darstellung

Abbildung 30: Evaluierung Messung Iframe

Name	Status	Typ	Initiator	Größe	Zeit	Wasserfall
EvalFrame	200	document	Andere	860 B	3 ms	
polyfills.js	200	script	EvalFrame	174 kB	5 ms	
styles.js	200	script	EvalFrame	127 kB	9 ms	
main.js	200	script	EvalFrame	129 kB	7 ms	
styles.css	200	stylesheet	EvalFrame	279 B	10 ms	
859.js	200	script	main.js:1	200 kB	5 ms	
32.js	200	script	main.js:1	57.2 kB	7 ms	
603.js	200	script	main.js:1	93.5 kB	9 ms	
644.js	200	script	main.js:1	22.4 kB	13 ms	
runtime.7f0554fa852fd0112.js	200	script	(Index)	1.4 kB	26 ms	
polyfills.d931bc7a3efc91cf.js	200	script	(Index)	37.5 kB	30 ms	
main.4fae66d0ab31add8.js	200	script	(Index)	140 kB	81 ms	
styles.24f1a699fa2da1a6.css	200	stylesheet	(Index)	74.0 kB	66 ms	
KFOICnqEu92Fr1Mu4nxKKTU1Kg.woff2	200	font	(Index)	11.1 kB	23 ms	
styles.24f1a699fa2da1a6.css	200	stylesheet	(Index):10	74.0 kB	24 ms	
ExamplePic.webp	200	webp	main.4fae66d0ab31add...	1.6 MB	208 ms	
KFOICnqEu92Fr1MmEU9fBbC4AMP6IQ.woff2	200	font	(Index)	11.1 kB	22 ms	

Quelle: Eigene Darstellung

Abbildung 31: Evaluierung Messung Web Component

Name	Status	Typ	Initiator	Größe	Zeit	Wasserfall
EvalWC	200	document	Andere	860 B	6 ms	
polyfills.js	200	script	EvalWC	174 kB	7 ms	
styles.js	200	script	EvalWC	127 kB	7 ms	
main.js	200	script	EvalWC	129 kB	11 ms	
styles.css	200	stylesheet	EvalWC	279 B	10 ms	
remoteEntry.js	200	script	main.js:1	6.7 kB	43 ms	
859.js	200	script	main.js:1	200 kB	8 ms	
32.js	200	script	main.js:1	57.2 kB	9 ms	
603.js	200	script	main.js:1	93.5 kB	11 ms	
644.js	200	script	main.js:1	22.4 kB	11 ms	
company-logo.png	200	document		10.0 kB	40 ms	
webcomponent.js	200	script	644.js:1	224 kB	43 ms	
ExamplePic.webp	200	webp	webcomponent.js:1	1.6 MB	17 ms	

Quelle: Eigene Darstellung

Abbildung 32: Evaluierung Messung Module Federation

Name	Status	Typ	Initiator	Größe	Zeit	Wasserfall
EvalMF	200	document	Andere	860 B	3 ms	
polyfills.js	200	script	EvalMF	174 kB	6 ms	
styles.js	200	script	EvalMF	127 kB	9 ms	
main.js	200	script	EvalMF	129 kB	7 ms	
styles.css	200	stylesheet	EvalMF	279 B	11 ms	
859.js	200	script	main.js:1	200 kB	5 ms	
32.js	200	script	main.js:1	57.2 kB	7 ms	
603.js	200	script	main.js:1	93.5 kB	9 ms	
644.js	200	script	main.js:1	22.4 kB	11 ms	
remoteEntry.js	200	script	main.js:1	204 kB	5 ms	
src_app_export_export_module_ts.js	200	script	load script:41	719 kB	8 ms	
ExamplePic.webp	200	webp	644.js:1	1.6 MB	26 ms	

Quelle: Eigene Darstellung

Abbildung 33: Evaluierung Messung Module Federation Web Components

Name	Status	Typ	Initiator	Größe	Zeit	Wasserfall
EvalMFWC	200	document	Andere	860 B	4 ms	
polyfills.js	200	script	EvalMFWC	174 kB	6 ms	
styles.js	200	script	EvalMFWC	127 kB	10 ms	
main.js	200	script	EvalMFWC	129 kB	7 ms	
styles.css	200	stylesheet	EvalMFWC	279 B	11 ms	
859.js	200	script	main.js:1	200 kB	11 ms	
32.js	200	script	main.js:1	57.2 kB	9 ms	
603.js	200	script	main.js:1	93.5 kB	10 ms	
644.js	200	script	main.js:1	22.4 kB	9 ms	
remoteEntry.js	200	script	main.js:1	205 kB	6 ms	
src_bootstrap_ts.js	200	script	load script:41	804 kB	8 ms	
ExamplePic.webp	200	webp	platform-browser.mjs:599	1.6 MB	9 ms	

Quelle: Eigene Darstellung

Abbildung 34: HTML eines eingebundenen Iframes

```

▼<iframe id="inlineFrameExample" title="Inline Frame Example" width="300" height="200" src="https://www.openstreetmap.org/export/embed.html?bbox=-0.00401794910430...47612752641776%2C0.0003057...%2C51.478569861898606&layer=mapnik">
  ▼#document
    <!DOCTYPE html>
    ▼<html xmlns="http://www.w3.org/1999/xhtml">
      ▶<head>...</head>
      ▼<body>
        ▶<div id="map" class="leaflet-container leaflet-touch leaflet-fade-anim leaflet-grab leaflet-touch-drag leaflet-touch-zoom" tabindex="0" style="position: relative;">...</div>
      </body>
    </html>
  </iframe>

```

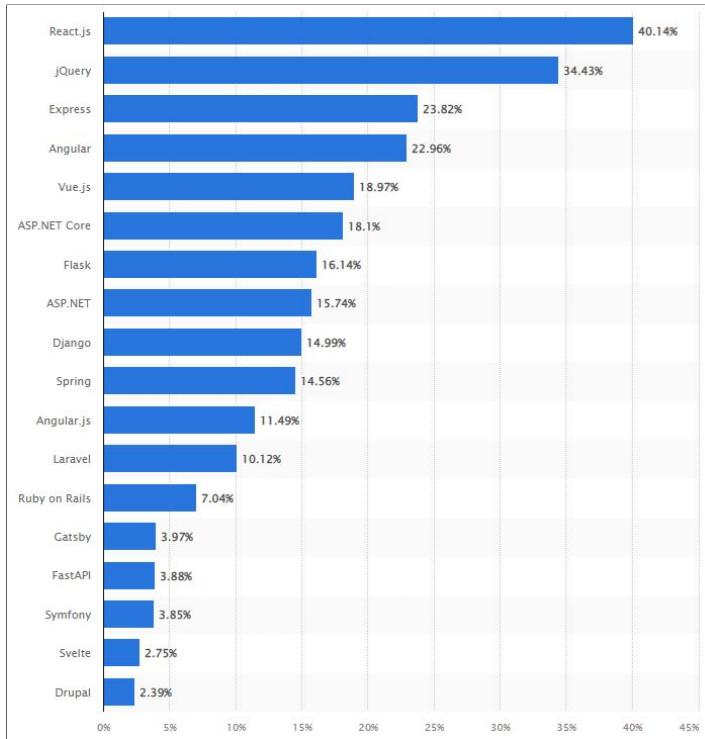
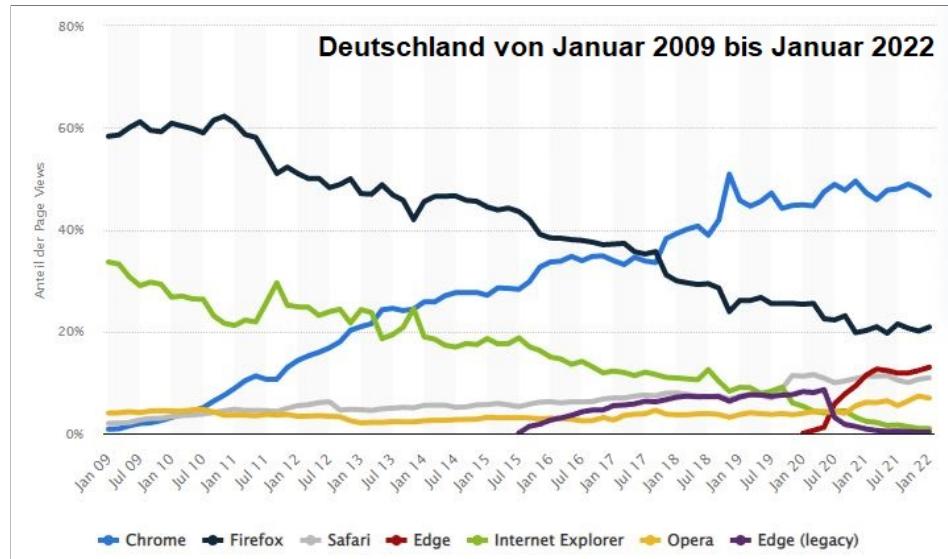
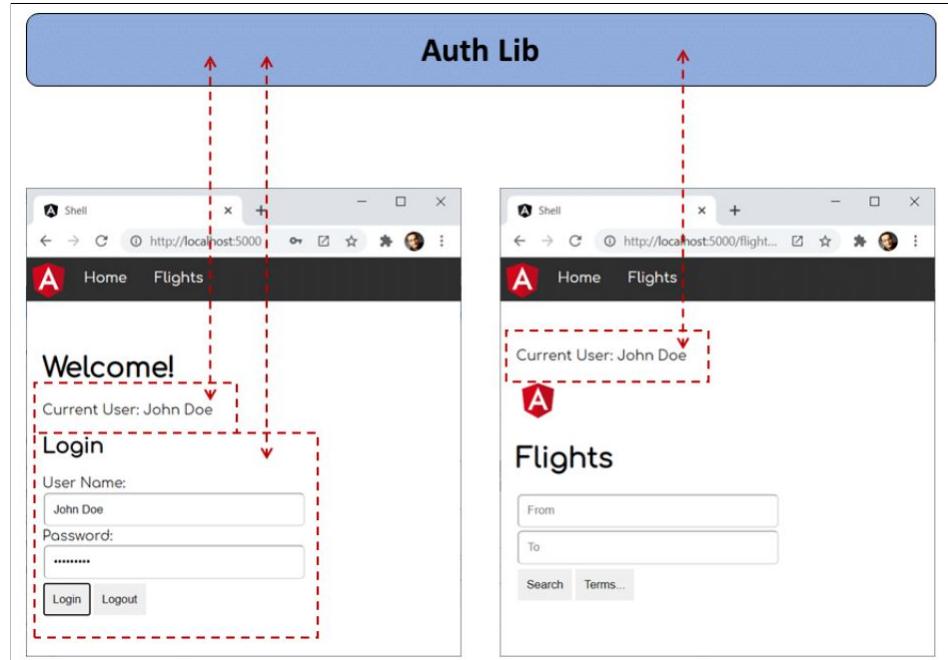
Quelle: Eigene Darstellung**Abbildung 35:** Populärste Web Frameworks 2021**Quelle:** Statista2022b

Abbildung 36: Marktanteile der führenden Browser



Quelle: Statista2022

Abbildung 37: Teilen von Daten bei Module Federation

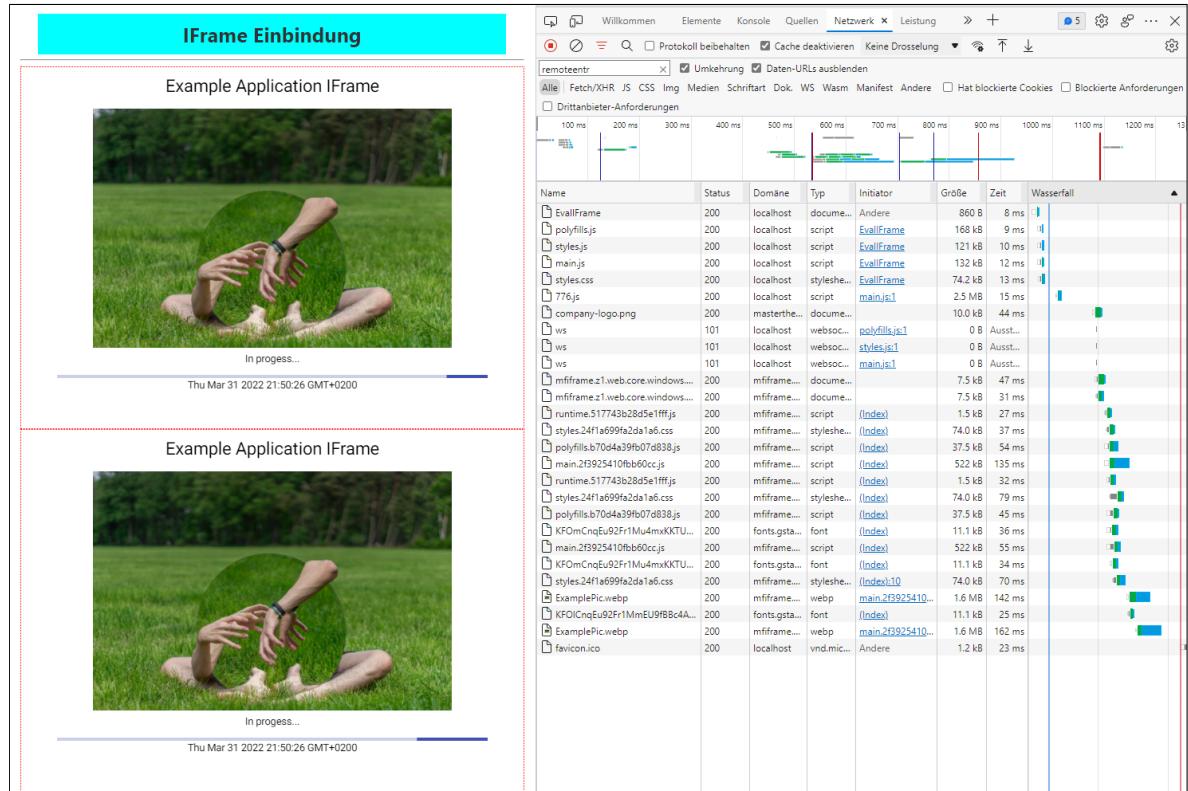


Quelle: Steyer2020

Abbildung 38: Durchgeführte Messungen mit durchschnittlicher Datenmenge

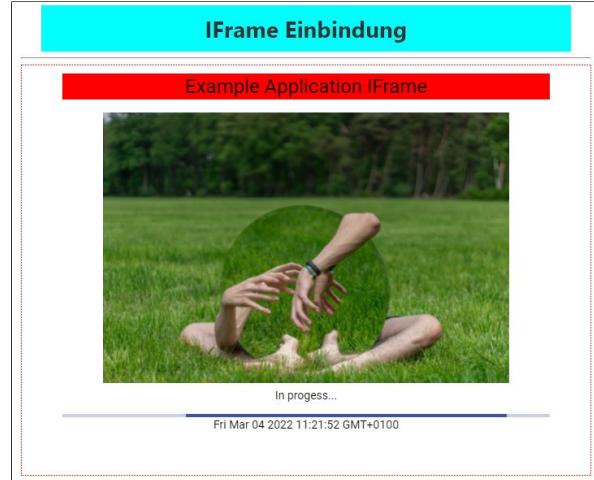
Anzahl		1		3		5		8	
Bibliotheken	Name	KB	Name	KB	Name	KB	Name	KB	
	ng core	234,9	ng core	234,9	cheerio	335	autoprefixer	820,1	
	corejs		corejs	194,4	moment	289,7	cheerio	335	
	jquery		jquery	87,9	ng core	234,9	moment	289,7	
					corejs	194,4	ng core	234,9	
					jquery	87,9	corejs	194,4	
					less		less	143,1	
					jquery		jquery	87,9	
					bluebird		bluebird	77,5	
Durchschnitt in KB	234,9		172,4		228,38		272,825		

Quelle: Eigene Darstellung

Abbildung 39: Doppelte Datenmenge durch zweifach eingebundenes Iframe

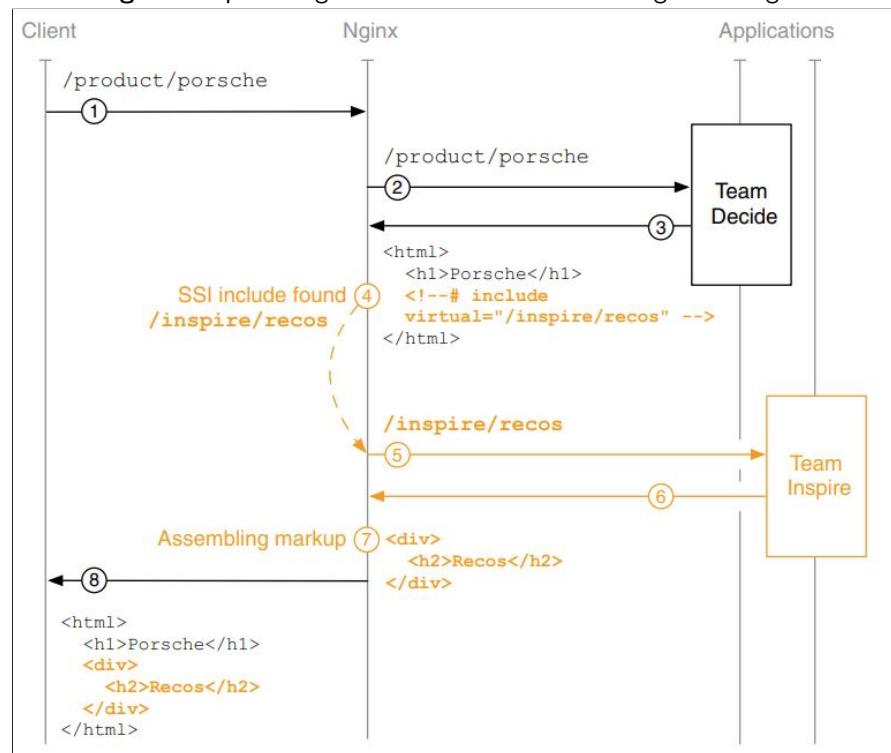
Quelle: Eigene Darstellung

Abbildung 40: Eingebundenes Iframe mit abgekapselten Stylings

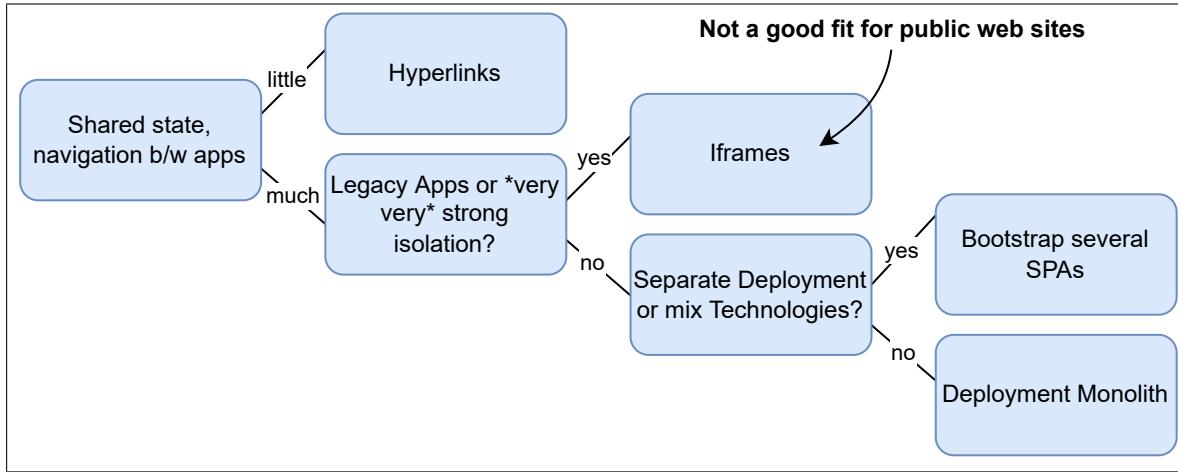


Quelle: Eigene Darstellung

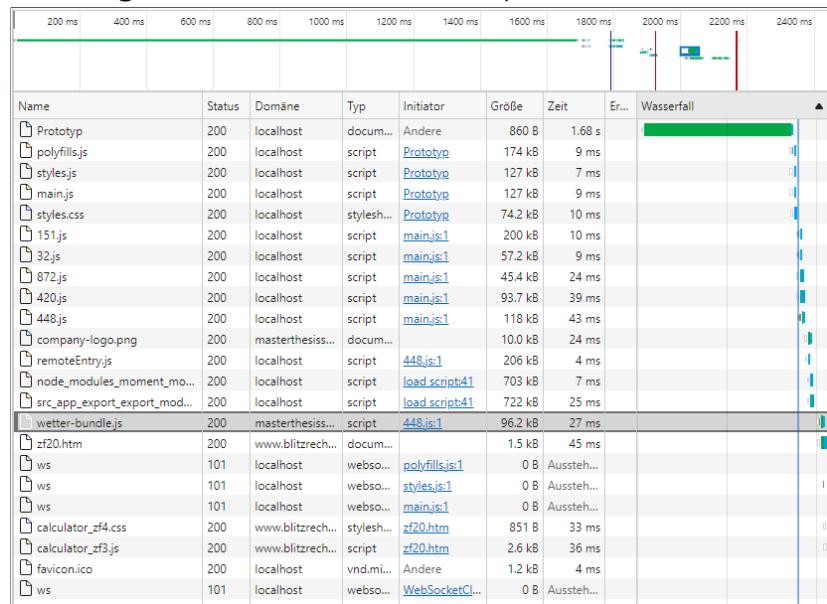
Abbildung 41: Sequenzdiagramm einer Zusammensetzung durch Nginx SSI



Quelle: Geers2020

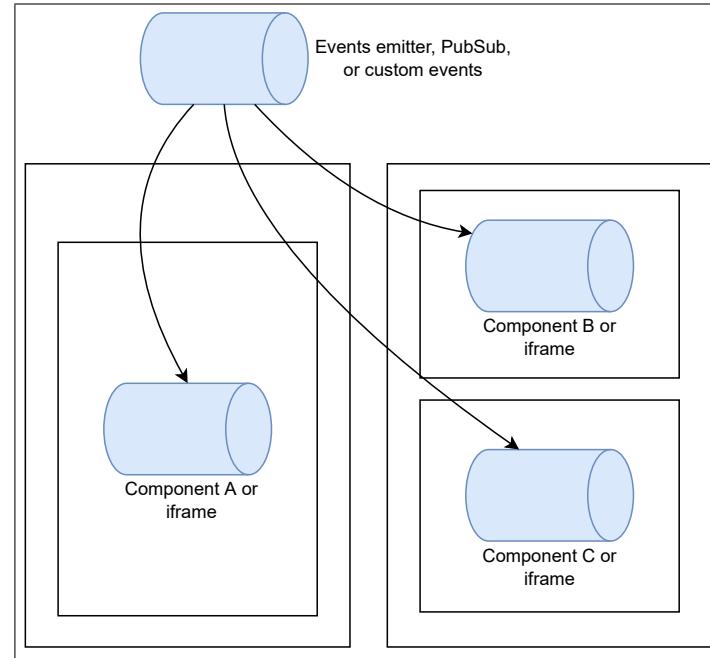
Abbildung 42: Entscheidungsbaum Technologie Microfrontends

Quelle: Steyer2018

Abbildung 43: Laden der Wetter Web Component

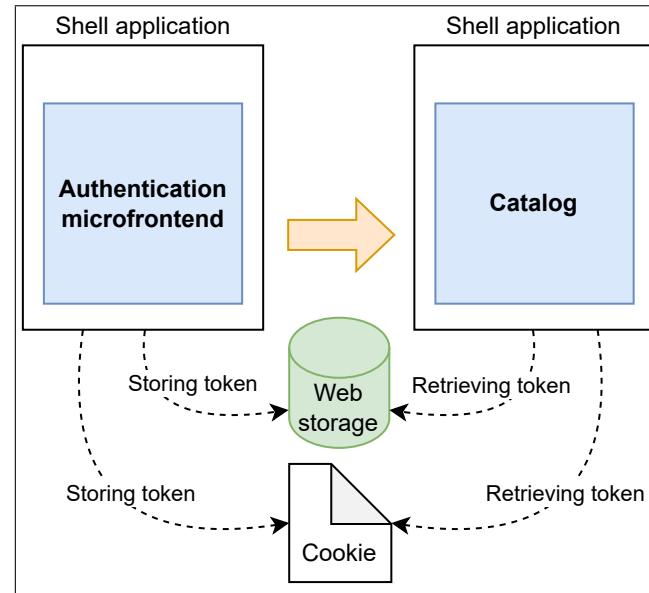
Quelle: Eigene Darstellung

Abbildung 44: Event Emitter zum Übertragen von Daten



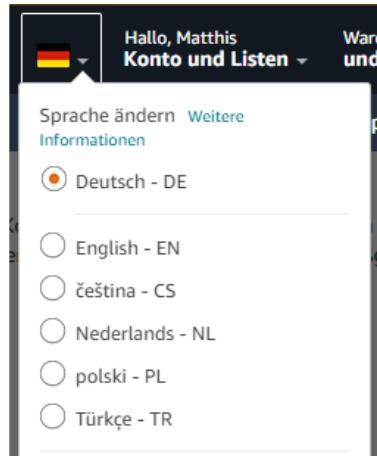
Quelle: Mezzalira2021

Abbildung 45: Web Storage zum Teilen von Daten



Quelle: Mezzalira2021

Abbildung 46: Beispielhafte realisierte Lokalisierung bei Amazon



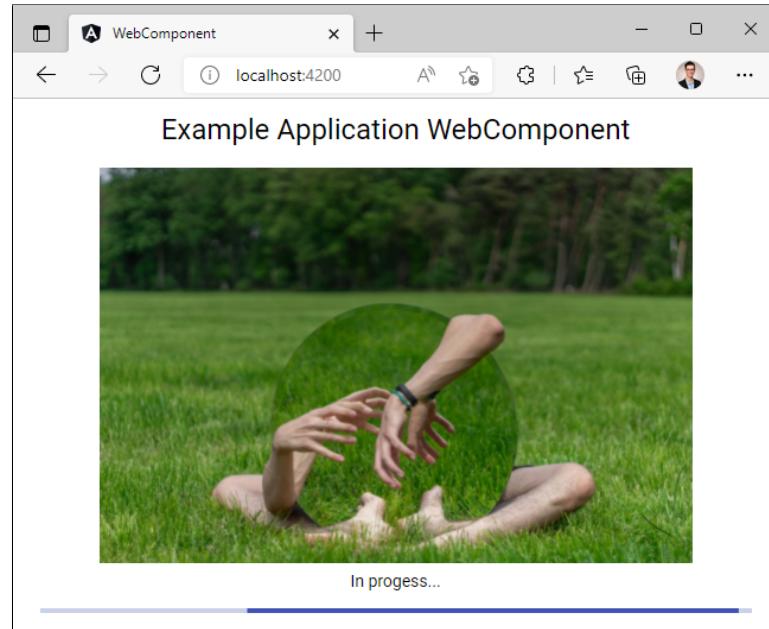
Quelle: Screenshot von Amazon

Abbildung 47: Zusammengefasste Präferenzmatrix mehrerer Teilnehmer

Kriterium	K1	K2	K3	Σ	Rang R	Inverser Rang I	Relatives Gewicht G
K1		0	1	1	3	1	0,1667
K2	3		3	6	1	3	0,5000
K3	2	0		2	2	2	0,3333
Summe				6		1	

Quelle: Schierenbeck2016

Abbildung 48: Web Component in der eigenen Anwendung eingebunden



Quelle: Eigene Darstellung

Abbildung 49: Module Federation WC Beeinflussung Styles



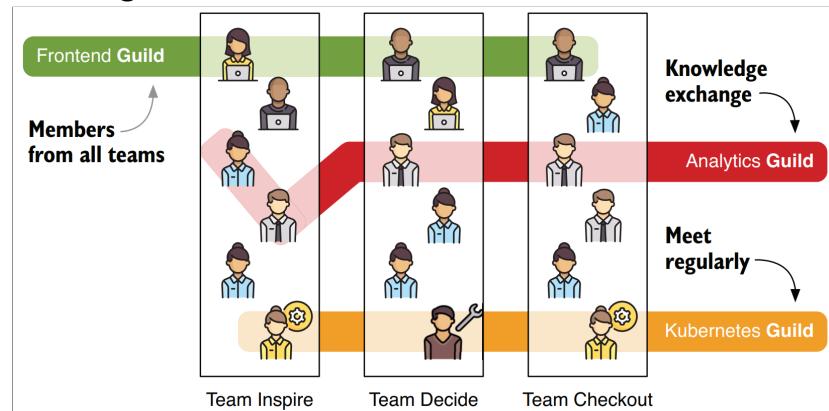
Quelle: Eigene Darstellung

Abbildung 50: Ladezeiten Module Federation WC bei 8 geteilten Bibliotheken

Name	Status	Typ	Initiator	Größe	Zeit	Wasserfall
ws	101	websocket	WebSocketClient:sc_16	0 B	Ausstehend	
EvalIMFWC	200	document	Andere	860 B	6 ms	
polyfills.js	200	script	EvalIMFWC	174 kB	6 ms	
styles.js	200	script	EvalIMFWC	126 kB	12 ms	
main.js	200	script	EvalIMFWC	137 kB	8 ms	
styles.css	200	stylesheet	EvalIMFWC	742 kB	14 ms	
remoteEntry.js	200	script	main.js[1]	6.7 kB	64 ms	
259.js	200	script	main.js[1]	89.6 kB	7 ms	
439.js	200	script	main.js[1]	381 kB	15 ms	
223.js	200	script	main.js[1]	200 kB	19 ms	
808.js	200	script	main.js[1]	57.2 kB	21 ms	
700.js	200	script	main.js[1]	93.7 kB	24 ms	
40.js	200	script	main.js[1]	903 kB	33 ms	
486.js	200	script	main.js[1]	205 kB	38 ms	
322.js	200	script	main.js[1]	146 kB	41 ms	
689.js	200	script	main.js[1]	353 kB	46 ms	
840.js	200	script	main.js[1]	803 kB	52 ms	
927.js	200	script	main.js[1]	227 kB	55 ms	
company-logo.png	200	document	Andere	100 kB	80 ms	
remoteEntry.js	200	script	main.js[1]	209 kB	5 ms	
ws	101	websocket	polyfills.js[1]	0 B	Ausstehend	
ws	101	websocket	styles.js[1]	0 B	Ausstehend	
ws	101	websocket	main.js[1]	0 B	Ausstehend	
src_bootstrap_ts.js	200	script	load script:41	807 kB	7 ms	
ExamplePic.webp	200	webp	platform-browser.mjs...	1.6 MB	19 ms	
favicon.ico	200	vnd.microsoft...	Andere	1.2 kB	4 ms	

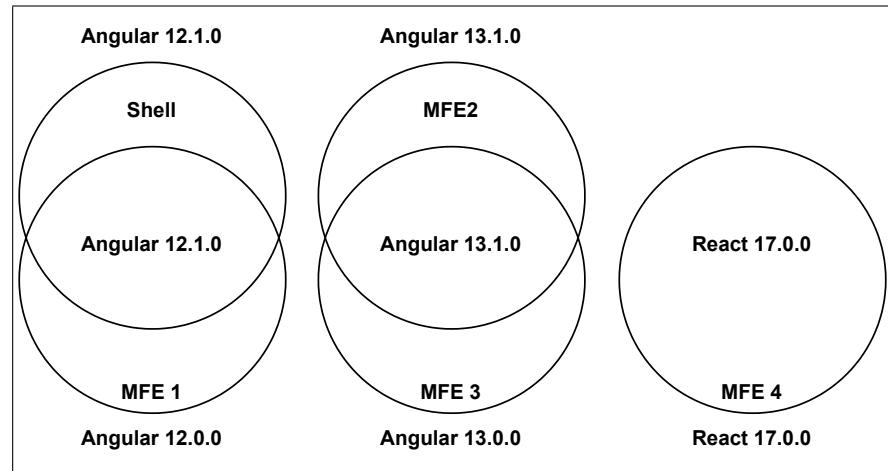
Quelle: Eigene Darstellung

Abbildung 51: Horizontaler Wissensaustausch in vertikalen Teams



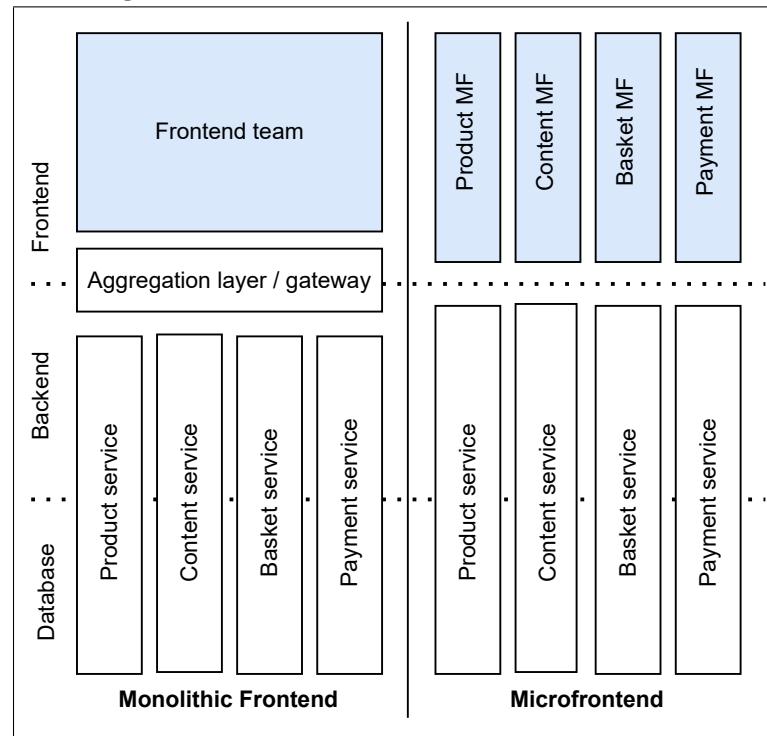
Quelle: Geers2020

Abbildung 52: Schnittmengen von Frameworks bei Module Federation



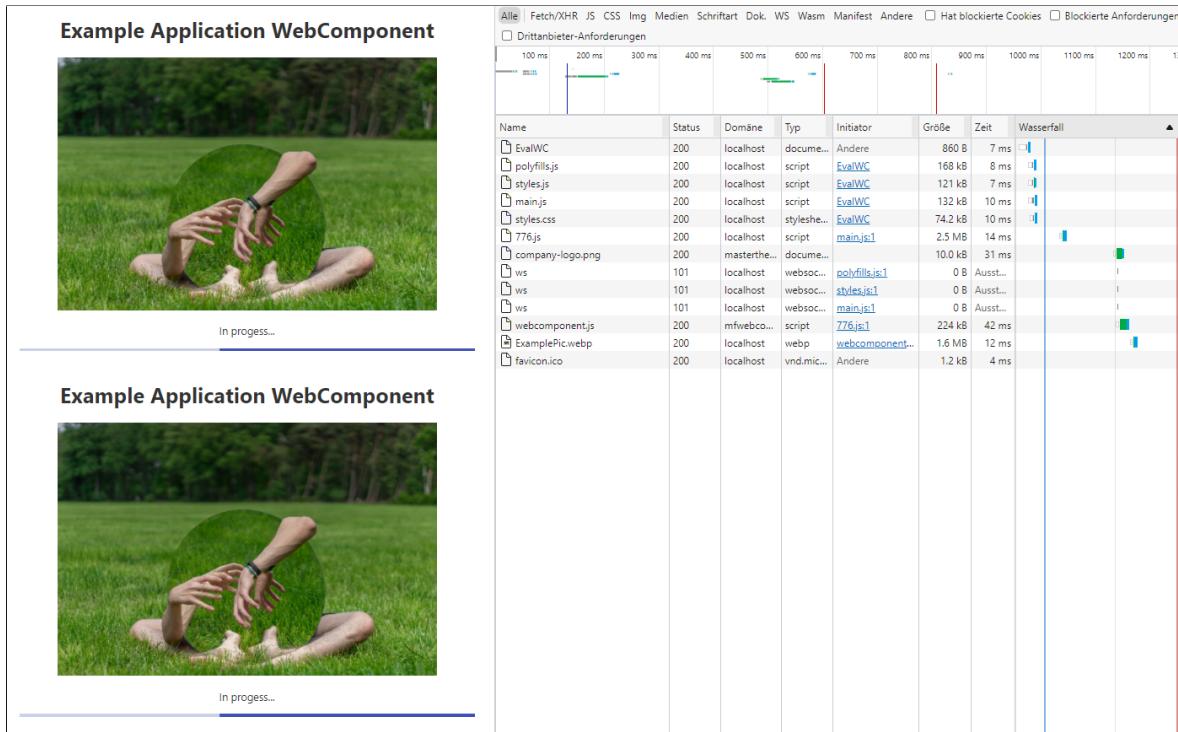
Quelle: Steyer 2021a

Abbildung 53: Monolithisches Frontend vs. Microfrontend



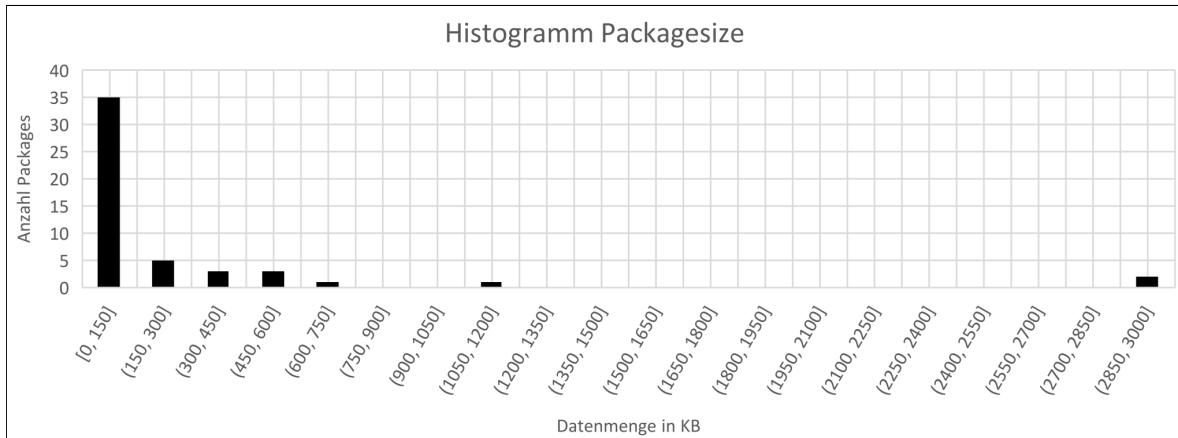
Quelle: Eigene Darstellung

Abbildung 54: Skalierung Web Component doppelte Einbindung

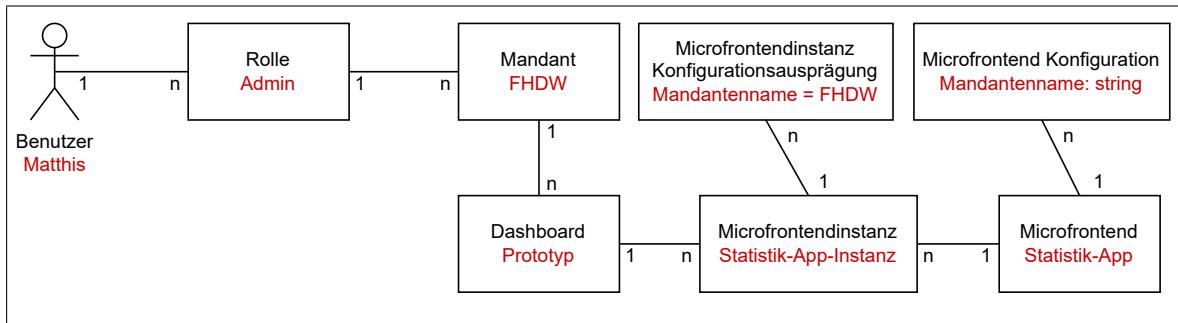


Quelle: Eigene Darstellung

Abbildung 55: Histogramm der Datenmenge von NPM-Paketen



Quelle: Eigene Darstellung

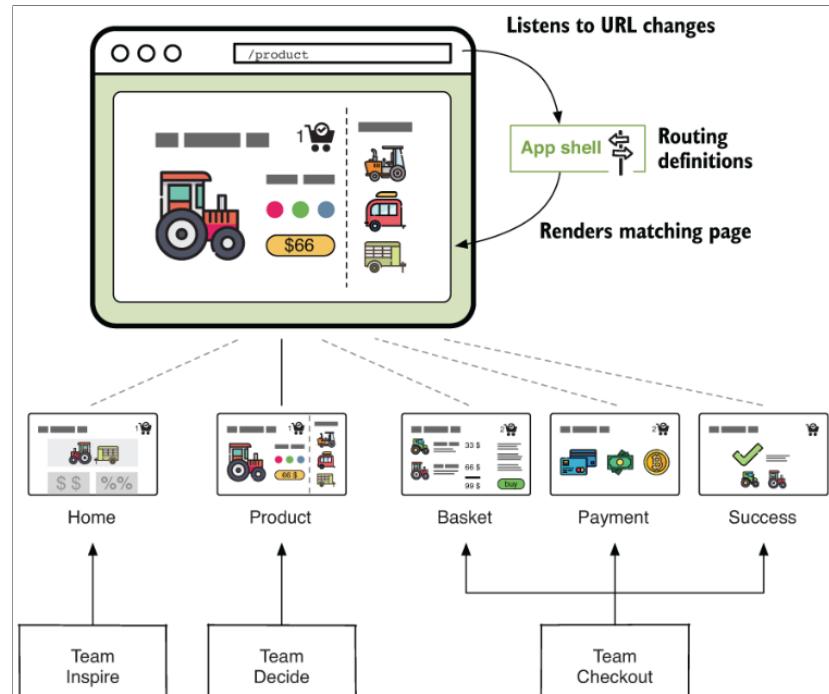
Abbildung 56: Übersicht der Elemente der Portalapplikation mit Beispielen

Quelle: Eigene Darstellung

Abbildung 57: Laborsetup der Messungen

Laborsetup	
Browser	Microsoft Edge
Version	100.0.1185.39 (Offizielles Build) (64-Bit)
Hostingumgebung	localhost
CPU	AMD Ryzen 7 3700X 8-Core Processor 3.60 GHz
RAM	32 GB
GPU	AMD Radeon RX 5700 XT
Anzahl Versuche	3 (je Art)
Gewähltes Ergebnis	Mittleres, nach Sortieren (nach Kriterium)
Angular Version	13.2.3

Quelle: Eigene Darstellung

Abbildung 58: Funktionsweise einer Portalapplikation

Quelle: Geers2020

Abbildung 59: Microfrontend mit *moment.js* geteilt durch Module Federation

Name	Status	Typ	Initiator	Größe	Zeit	Wasserfall
EvalMF	200	document	Andere	860 B	331 ms	I
polyfills.js	200	script	EvalMF	174 kB	5 ms	
styles.js	200	script	EvalMF	127 kB	10 ms	
main.js	200	script	EvalMF	129 kB	7 ms	
styles.css	200	stylesheet	EvalMF	74.2 kB	12 ms	
439.js	200	script	main.js:1	381 kB	6 ms	
223.js	200	script	main.js:1	200 kB	8 ms	
808.js	200	script	main.js:1	57.2 kB	10 ms	
700.js	200	script	main.js:1	93.5 kB	12 ms	
927.js	200	script	main.js:1	22.7 kB	15 ms	
remoteEntry.js	200	script	main.js:1	205 kB	313 ms	I
src_app_export_export_module_ts.js	200	script	load script:41	719 kB	11 ms	
ExamplePic.webp	200	webp	927.js:1	1.6 MB	9 ms	

Quelle: Eigene Darstellung

Anhang 3 Tabellen

Tabelle 7: Verwendete NPM-Pakete zum Vergleich der Datenmenge

Name	Datenmenge in KB (Stand 28.02.2022)
autoprefixer	820.1
cheerio	335
moment	289.7
@angular/core	234,9
corejs	194.4
less	143.1
jquery	87.9
bluebird	77.5
Durchschnitt	272.8

Quelle: Eigene Darstellung

Tabelle 8: Übersicht der Rollen der prototypischen Portalshell

ID	Rollenname	Beschreibung
1	Anwender	Der Anwender darf die Portalapplikation benutzen, aber nichts konfigurieren. Er sieht die Dashboards seines Mandanten und kann die Microfrontendinstanzen in den Dashboards benutzen.
2	Administrator	Der Administrator darf die Portalapplikation sowie die Microfrontendinstanzen vollumfänglich nutzen. Er kann neue Benutzer registrieren und ihnen Rollen zuweisen. Ebenfalls sieht der Administrator eventuell mehr Inhalte in den Microfrontendinstanzen, sollte es dort Bereiche geben, die ausschließlich für Administratoren zugänglich sind. Der Administrator ist ebenfalls in der Lage Microfrontendinstanzen auf einem Dashboard zu platzieren und neu anzugeordnen.

Quelle: Eigene Darstellung

Anhang 4 Expertengespräche

Exp 1

Expertengespräch mit Hanno Kortekamp

Gesprächsprotokoll mit Hanno Kortekamp, Senior Software-Architekt bei einem großen deutschen IT-Dienstleister, mit mehr als 10 Jahren Berufserfahrung in Entwicklung und Betrieb von Portalapplikationen.

Durchgeführt am 20.12.2021 zu den Themen: Microfrontends, Portalapplikationen und Module Federation.

Querschnittsaspekte von Portalapplikationen

- SDK für Formularelemente, Theme, UI und Styleguide
- Authentifizierung und Autorisierung
- Konfiguration
- Lokalisierung
- Navigation
- Mandantenfähigkeit

Messkriterien für Integration Microfrontends

- Übertragene Datenmenge
- Renderingzeit bis Darstellung
- Entwicklungsaufwand / Wartbarkeit
- Nähe am Standard
- Unterstützung verschiedener Frameworks
- (Browser)Kompatibilität
- Autonomie
- Unabhängigkeit der Entwicklerteams
- Lock-In Effekt

Einsatz Module Federation im Projekt

- Nur Module Federation beschränkt Unabhängigkeit der Entwicklerteams
- Web Components in Kombination mit Module Federation mehr flexibel

Exp 2

Expertengespräch im Rahmen der Nutzwertanalyse

Diskussion im Rahmen der Nutzwertanalyse. Teilnehmerkreis bestehend aus:

Zwei Software-Architekten, zwei Software-Entwicklern und einem IT-Projektmanager. Alle Teilnehmer haben bereits mehrjährige Erfahrungen mit Portalapplikationen und Microfrontends sammeln können.

Durchgeführt am 03.02.2022 zu den Themen: Vorstellung & Diskussion gesammelter Kriterien sowie anschließende individuelle Durchführung der Paarvergleichsmethode.

Gesammelte Kriterien:

- Datenmenge
- Renderingzeit
- Entwicklungsaufwand
- Wartbarkeit
- Nähe zum Standard
- Frameworks
- Kompatibilität
- Autonomie
- Unabhängige Entwicklerteams
- Lock-In Effekt

Sowie Ergänzung des Kriteriums *Interoperabilität*.

Anschließend individuelle Durchführung der Paarvergleichsmethode. Die einzelnen Paarvergleichsmatrizen sind dem beigefügten Anhang 5 enthalten und in der *Paarvergleichsmatrizen.zip* zu finden.

Anhang 5 Sonstige Dokumente

Hinweis: Die sonstigen Dokumente sind nur in der beigefügten zip-Datei enthalten.

1. Internetquellen
2. Tabelle Nutzwertanalyse
3. Tabellen Paarvergleichsmatrixen
4. Tabelle Berechnung übertragene Datenmenge
5. Messergebnisse geteilte Bibliotheken Web Components
6. Quellcode Microfrontends
7. Quellcode Portalshell inkl. StatistikApp

Quellenverzeichnis

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterthesis selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Gütersloh, 21. April 2022

Matthis Wieneke