



Abschlussprüfung Sommer 2022

Fachinformatiker für Anwendungsentwicklung  
Dokumentation zur betrieblichen Projektarbeit

# **Fußzeile als Komponente mit funktionalen Knöpfen und Darstellung von Informationen für die Abschlussstrecke eines Versicherungsproduktes**

Abgabetermin: Hannover, den 27.04.2022

Prüfungsausschuss: FIAE Han 3 103

**Prüfungsbewerber:**

Prüflingsnummer: 94023

Jakob Schumann

Goethestraße 40

30169 Hannover



**Ausbildungsbetrieb:**

Konzept & Marketing – ihr unabhängiger Konzeptentwickler GmbH

Podbielskistraße 333

30659 Hannover

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Projektumfeld . . . . .	1
1.2. Projektziel . . . . .	1
1.3. Projektbegründung . . . . .	2
1.4. Projektschnittstellen . . . . .	3
1.5. Projektabgrenzung . . . . .	3
<b>2. Projektplanung</b>	<b>3</b>
2.1. Projektphasen . . . . .	4
2.2. Abweichung vom Projektantrag . . . . .	4
2.3. Ressourcenplanung . . . . .	4
2.4. Entwicklungsprozess . . . . .	5
<b>3. Analysephase</b>	<b>5</b>
3.1. Ist-Analyse . . . . .	5
3.2. 'Make, Buy, or Reuse' - Analyse . . . . .	6
3.3. Projektkosten . . . . .	7
3.4. Amortisationsrechnung . . . . .	8
<b>4. Entwurfsphase</b>	<b>8</b>
4.1. Zielplattform . . . . .	8
4.2. Architekturdesign . . . . .	8
4.3. Entwurf der Benutzeroberfläche . . . . .	9
4.4. Entwurf der Datenstruktur . . . . .	9
4.5. Entwurf der Geschäftslogik . . . . .	10
4.6. Maßnahmen zur Qualitätssicherung . . . . .	10
<b>5. Implementierungsphase</b>	<b>11</b>
5.1. Implementierung der Datenstruktur . . . . .	11
5.2. Implementierung der Benutzeroberfläche und Frontendlogik . . . . .	11
5.3. Implementierung der Geschäftslogik . . . . .	12
<b>6. Projektabschluss</b>	<b>13</b>
6.1. Abnahme . . . . .	13
6.2. Soll-Ist-Vergleich . . . . .	13
6.3. Gewonnene Erkenntnisse . . . . .	14
6.4. Ausblick . . . . .	14
<b>Literaturverzeichnis</b>	<b>I</b>
<b>Abkürzungsverzeichnis</b>	<b>II</b>

<b>Glossar</b>	<b>III</b>
<b>A. Anhänge</b>	<b>IV</b>
A.1. Anwendungsfalldiagramm . . . . .	IV
A.2. Oberflächenabbildungen . . . . .	IV

## Tabellenverzeichnis

1. Gebrauchte Zeit in Stunden aufgeteilt in Projektphasen . . . . .	4
2. Nutzwertanalyse: Vergleich Onlinetarifrechner (OTR) . . . . .	6
3. Vergleich gebrauchte Zeit der Projektphasen . . . . .	13

# 1. Einleitung

## 1.1. Projektumfeld

### **Konzept und Marketing GmbH**

Bei der Konzept und Marketing GmbH (K&M) handelt es sich um einen Finanzdienstleister der Versicherungskonzepte entwickelt. Die Konzepte werden über Partnerschaften mit namhaften Versicherern auf den Markt gebracht. K&M übernimmt die vollständige Verwaltung, vom Vertrieb durch freie Makler/-innen über die Vertragsannahme, bis zur Schadenregulierung. Das Unternehmen, bestehend derzeit aus 118 Mitarbeiter/-innen, bietet Dienstleistungen sowie Portale für Versicherer, Makler/-innen und Kunden/Kundinnen an.

### **Mariosoft**

Bei der Mariosoft handelt es sich um eine IT-Abteilung der K&M. Die Mariosoft besteht derzeit aus 18 Mitarbeiter/-innen und beschäftigt sich mit der Konzeption, Programmierung und Wartung von internen Softwarelösungen wie Onlinetarifrechner (OTR), Kundenverwaltung und Übersichtsseiten für Makler. Das umzusetzende Projekt ist im Umfeld der Mariosoft angesiedelt und ist eine Komponente für eine gleichzeitig entstehende Webanwendung.

## 1.2. Projektziel

Ziel des Gesamtprojekts ist ein neuer, modularer OTR als Webseite bzw. Single-Page-Application (SPA). Dieser soll so entworfen werden, dass die einzelnen Komponenten austauschbar sind und für weitere Versicherungsprodukte benutzt werden können. Außerdem soll die Benutzeroberfläche einheitlich gestaltet werden. Allgemein soll die Einführung neuer und die Erweiterung bestehender Versicherungsprodukte vereinfacht werden. Der OTR wird aus einem .NET 6.0 Backend, welches Schnittstellen zu umliegenden firmeninternen Systemen hat, bestehen. Des Weiteren ist ein Cachespeicher und weitere Logik vorhanden. Das Frontend besteht aus Vue.js<sup>1</sup> mit Designvorlagen von Vuetify<sup>2</sup> sowie einer eigenen Oberflächenkomponentenbibliothek, welche Vuetifyelemente kapselt und im Corporate-Design<sup>3</sup> zur Verfügung stellt. Im Frontend wird TypeScript als Programmiersprache verwendet.

Als Versionsverwaltung für dieses Projekt wird Git mit GitFlow genutzt. Das Continuous Integration/Continuous Delivery (CI/CD) wird mittels GitLab Pipelines realisiert, eine Möglichkeit automatisch nach jeder Änderung das Projekt zu bauen, zu testen und zu veröffentlichen. Veröffentlicht wird der OTR in einem Kubernetes-Cluster. Im Abschnitt 4.1 Zielplattform wird auf das Verfahren weiter eingegangen.

Bei einem OTR haben Makler/-innen die Möglichkeit Beiträge für versicherte Risiken zu errechnen. Darüber hinaus sind hilfreiche oder sogar verpflichtende Funktionalitäten gegeben, wie z.B. das Ausdrucken von Anträgen und Angeboten, das Speichern von Angeboten, um diese später wieder laden und weiter bearbeiten zu können und das Versenden von tarifrelevanten Dokumenten per E-Mail. Diese Funktionalitäten werden in der Fußzeile in Form von Schaltflächen angeboten. Die Implementierung dieser Fußzeile wird im Rahmen dieses Projekts umgesetzt. Die Fußzeile beinhaltet außerdem Angaben zum Copyright und einen

---

<sup>1</sup>s.[1] Vue.js

<sup>2</sup>s.[2] Vuetify

<sup>3</sup>s.[3] Corporate Design, Wikipedia

Verweis zum Impressum.

Beim Speichern eines Angebots sollen alle im OTR eingegebenen Daten im sogenannten Maklerportal für die angemeldeten Makler/-innen gespeichert werden. Diese können über das Maklerportal wieder geladen werden. Des Weiteren ist vorgesehen, dass Makler/-innen Angebote kopieren können. Die K&M bietet auch OTR außerhalb des Maklerportals an. Für diese Rechner ist keine Angebots- oder Kopiespeicherung vorgesehen, dementsprechend sind die Schaltflächen dort nicht vorhanden.

Die Druckenfunktionalität wird durch zwei Schaltflächen abgebildet, über die erste kann ein Angebot und über die zweite ein Antrag ausgedruckt werden. Ein Angebot wird einem Kunden oder einer Kundin von Makler/-innen unterbreitet und ist, wie der Antrag, ein verbindliches Dokument mit welchem ein Versicherungsvertrag abgeschlossen werden kann. Die Funktionalität wird häufig genutzt und ist damit wichtig. Die Erzeugung von Angebots- und Antragsdokumenten setzt die Berechnung der Risiken voraus. Dafür müssen alle berechnungsrelevanten Angaben getroffen worden sein. Nur bei einer gültigen Berechnung werden die Schaltflächen aktiv. Außerdem kann bei einem Tarif der Angebots- oder Antragsdruck deaktiviert sein. Dementsprechend sind die Schaltflächen aktiviert oder deaktiviert. Drucken bedeutet in diesem Zusammenhang das Herunterladen eines PDF-Dokumentes. Das Dokument wird von einem angebundenem System anhand der Daten im XML<sup>4</sup>-Format generiert.

Eine weitere Funktion ist die des Dokumentenversands. Beim Betätigen dieser Schaltfläche öffnet sich ein Dialog, in dem der Nutzer die gewünschten Dokumente aus allen relevanten Dokumenten auswählen kann um sie zu versenden. Dazu gehören auch die zuvor beschriebenen Angebots- und Antragsdokumente. Zusätzlich müssen eine oder mehrere E-Mailadressen angegeben werden, damit die Dokumente an diese E-Mailadressen in einem ZIP-Archiv versendet werden können. Die Dokumente beinhalten rechtliche Informationen und den Antrag sowie das Angebot. Die komplette Dokumentenversand-Komponente liegt als eigenständige Bibliothek vor, da sie auch für andere OTR genutzt wird, so dass diese für das Frontend eingebunden werden musste.

Die beschriebenen Schaltflächen sind als Anwendungsfälle in einem Anwendungsfalldiagramm (s. A.1 Anwendungsfalldiagramm der Fußzeilenkomponente) dargestellt.

### **1.3. Projektbegründung**

Die Motivation hinter diesem Projekt begründet sich in der einfachen Erweiterbarkeit für neue Versicherungsprodukte sowie ein Redesign des Aussehens des OTR. Zusätzlich ist der OTR durch den modularen und komponentenbasierten Aufbau des Projekts besser wartbar.

Die Fußzeile ist eine Komponente die in allen bestehenden Produkten und zukünftig kommenden Produkten vorhanden sein muss. Die Platzierung der gegebenen Funktionalitäten in der Fußzeile bietet sich dafür gut an, da sie unabhängig von der übrigen Webseite funktionieren soll.

---

<sup>4</sup>Vgl.[4] Extensible Markup Language, Wikipedia

## 1.4. Projektschnittstellen

Das Backend weist Schnittstellen zu einigen Systemen auf. Es existiert das Tarifrechner-Gateway (TG), welches den Einstiegspunkt von Extern zu den OTR darstellt. Dieses sowie der OTR benutzt das Tarifrechner-REST-Gateway (TRG), welches als REST-Schnittstelle funktioniert um unter anderem Konfigurationen für Tarife oder das Speichern von Angeboten anzubieten. Eine weitere Schnittstelle die von der Fußzeilenkomponente genutzt wird, ist der PDFToolsservice, welcher die Generierung von PDF-Dokumenten übernimmt. Die Fußzeile sowie alle Komponenten im Frontend bekommen ihre Daten über den LocalStorage des Browsers (Vue.js Pinia-Store<sup>5</sup>) geliefert.

Das Maklerportal ist eine Webseite an der sich Makler/-innen anmelden und den OTR aufrufen können, Übersichten über ihren Kundenbestand haben und noch vieles mehr. Außerdem steuert das Maklerportal die Konfiguration der OTR. Zudem können dort gespeicherte Angebote eingesehen, bearbeitet und mit einem OTR aufgerufen werden.

Der OTR wird überwiegend von Makler/-innen benutzt um Tarife für ihre Kunden oder Kundinnen auszurechnen.

## 1.5. Projektabgrenzung

Das bearbeitete Projekt ist hierbei die Fußzeile mit den Schaltflächen einschließlich ihrer Funktionalitäten. Eine Benutzeroberflächenkomponentenbibliothek ist gegeben, aus welcher die Schaltflächen eingebunden werden. Ebenso ist eine Synchronisation zwischen Front- und Backend durch HTTP-JSON-PATCH schon vorhanden. Weiterhin ist ein Backend mit mehreren Schnittstellen, welche für die einzelnen Funktionalitäten erweitert werden müssen, verfügbar. Im Frontend ist ein Gridlayout mit Elementvorlagen gegeben, welche ebenfalls ausgebaut werden müssen.

## 2. Projektplanung

Das Projekt wird in dem Zeitraum vom 01.03.2022 bis zum 18.03.2022 durchgeführt.

---

<sup>5</sup>s.[5] What is Pinia?

## 2.1. Projektphasen

Projektphasen	Zeit in Stunden
Tägliche Absprachen mit Entwicklungsteam & Auftraggeber	1
Wöchentliche Absprache zur Oberfläche	1
<b>Ist-Analyse:</b>	<b>6</b>
Analyse vorhandener Codebasen und Funktionalitäten	3
Gespräche mit dem Auftragsgeber	3
<b>Sollkonzept:</b>	<b>4</b>
Planung des Soll-Zustands Projekt allgemein	2
Planung des Soll-Zustands Fußzeilenfunktionalität	1
Abstimmung & Sichtung Workflows und Oberflächenentwürfe	1
<b>Realisierung:</b>	<b>30</b>
<b>Frontend</b>	<b>13</b>
Schaltflächeneinbindung & Styling	3
Funktionalität Backend Anbindung, abhängiges Rendering	7
Eventhandling, Texte mit Internationalisierung	2
Entwicklung Oberfläche	1
<b>Backend</b>	<b>17</b>
Angebotsspeicherung	8
Angebotskopiespeicherung	2
Dokumentengenerierung	5
Dokumentenversand	2
<b>Dokumentation</b>	<b>19</b>
<b>Testen &amp; Abnahme</b>	<b>8</b>
<b>Gesamt</b>	<b>69</b>

Tabelle 1: Gebrauchte Zeit in Stunden aufgeteilt in Projektphasen

## 2.2. Abweichung vom Projektantrag

Alle im Projektantrag beschriebenen Funktionalitäten werden umgesetzt. Hinzu kommt die Kopiespeicherung eines Angebots. Außerdem ist schon ein Grundgerüst in Form von bestehenden Serviceprojekten und Schnittstellen gegeben. Die UI-Elemente müssen teilweise neu erstellt, erweitert oder angepasst werden. Dadurch, dass Mockups für die Benutzeroberfläche gegeben sind, muss weniger Zeit in die Konzeption der Benutzeroberfläche fließen. Deswegen weichen die angegebenen Zeiten aus Abschnitt 2.1 Projektphasen leicht von den Zeiten aus dem Projektantrag ab (s. 6.2 Soll-Ist-Vergleich).

## 2.3. Ressourcenplanung

Für die Umsetzung des Projekts wurden Ressourcen verwendet für die K&M bereits Lizenzen hat oder die unentgeltlich zur Verfügung stehen. Des Weiteren wurde Hardware verwendet die bereits im Besitz des Unternehmens ist. Folgende Ressourcen wurden verwendet:

- **Technische Ressourcen**

- Windows 10
- Visual Studio 2022
- Webstorm 2021
- TeXstudio
- GitLab
- Fork - GitClient
- Kubernetes/Docker
- Microsoft Teams für Besprechungen

- **Personen**

- Produktmanagement, für Rückfragen zum umzusetzenden Produkt
- Vertrieb
- externer Versicherer für Anforderungsdefinition
- Marketingabteilung für Designanforderungen
- Auszubildender für die Umsetzung und Tests

## 2.4. Entwicklungsprozess

Das Projekt wird mit dem Spiralmodell durchgeführt und durch ein Kanban-Board unterstützt. Vor dem Start der Bearbeitung wurden grobe Aufgaben und Anforderungen definiert. Es gibt tägliche Absprachen zwischen dem Entwicklungsteam sowie wöchentliche Absprachen mit dem Produktmanagement und der Marketingabteilung bezüglich der Oberfläche. Aus diesen Absprachen entstehen iterativ neue Anforderungen. Durch CI/CD werden inkrementell umgesetzte Funktionalitäten bzw. behobene Fehler zum Testen zur Verfügung gestellt.

Dieses Vorgehensmodell wurde gewählt damit die Entwicklung schnell an neue Anforderungen der Kunden/Kundinnen bzw. der Stakeholder angepasst werden kann.<sup>6</sup>

## 3. Analysephase

### 3.1. Ist-Analyse

Die K&M stellt den Maklern OTR zur Verfügung. Zur Zeit gibt es mehrere Generationen an OTR. Die älteste Generation ist mit PHP umgesetzt und hat keine moderne Benutzeroberfläche. Die bis heute neuste OTR Generation ist im Grundaufbau wie diese Neuentwicklung mit einem .NET-Backend und einem Vue.js Frontend aufgebaut. Da es sich um das erste Vue.js Projekt der Mariosoft handelte, gibt es jedoch architektonische Schwächen, die eine einfache Integration weiterer Versicherungsprodukte verhinderte.

Demnach existieren funktionale OTR, die beim Vertrieb von Versicherungsprodukten helfen, allerdings sind diese nicht intuitiv nutzbar und auf einem alten Corporate-Design aufgebaut sowie für Entwickler/-innen schwierig zu warten und zu erweitern.

---

<sup>6</sup>Vgl.[6] Spiralmodell: Das Modell für Risikomanagement in der Softwareentwicklung



### 3.2. 'Make, Buy, or Reuse' - Analyse

Dadurch, dass die K&M neue und innovative Versicherungskonzepte auf den Markt bringt, eignen sich sogenannte Baukastenrechner nicht für diese Tarife. Die Vorlagen dieser Rechner sind nicht so individuell gestaltbar wie die K&M es benötigt. So können z.B. individuelle Leistungen für die Endkunden/Endkundinnen nicht zur Auswahl gestellt werden, da diese in einem Baukastenrechner nicht umsetzbar wären. Zum Beispiel wird von der K&M eine Eigenheimversicherung in Kombination mit einer Hausratversicherung angeboten. Dieses Versicherungskonzept könnte bei marktüblichen Baukastenrechnern nur über mehrere Rechner kalkuliert werden.

Da es schon ältere Generationen an OTR gibt, wäre auch eine Überlegung eine ältere Onlinetarifrechnerbasis weiter zu benutzen anstatt einen neuen einzuführen. Die älteste Generation ist dagegen technologisch nicht tragbar und muss grundlegend erneuert werden. Der neuste OTR sollte die Grundlage werden, wiederum ist beim Umsetzen von weiteren Produkten und Leistungen aufgefallen, dass dieser architektonische Fehler hat, so dass dieser nicht geeignet ist und eine Neukonzeption schneller und günstiger als die Behebung der bestehenden Fehler ist.

In der folgenden Nutzwertanalyse wurden die verschiedenen OTR miteinander verglichen. Hierbei wurde sich für die Kriterien Aussehen, Aktualität der Technologie, bzw. die Zukunftssicherheit, Erweiterbarkeit und Anpassbarkeit entschieden. Erweiterbarkeit und Anpassbarkeit haben eine große Gewichtung, da dies für die K&M aufgrund individueller Tarife aber auch für die Entwicklung in der Mariosoft, sehr wichtig ist. Ebenso ist die Zukunftssicherheit und die Aktualität der eingesetzten Technologien wichtig. Weniger relevant ist das Aussehen der Benutzeroberfläche, trotzdem fließt diese in die Bewertung ein, da dieser Punkt für die Kunden und Kundinnen wichtig ist. Die Bewertung geht von 0-10 Punkten, wobei 10 die bestmögliche Bewertung ist.

Kriterium	Gewichtung	1. Generation OTR („Universal“)		2. Generation OTR („Spirit“)		3. Generation OTR	
		Bewertung	Bewertung (gewichtet)	Bewertung	Bewertung (gewichtet)	Bewertung	Bewertung (gewichtet)
Aussehen	15%	3	4,5	7	10,5	9	13,5
Technologie	25%	2	5	7	17,5	9	22,5
Erweiterbarkeit	30%	3	9	6	18	8	24
Anpassbarkeit	30%	3	9	5	15	7	21
<b>Summe</b>	<b>100%</b>	<b>11</b>	<b>27,5</b>	<b>25</b>	<b>61</b>	<b>33</b>	<b>81</b>

Tabelle 2: Nutzwertanalyse: Vergleich OTR

Der Rechner „Universal“ ist in PHP programmiert. Die Entwickler/-innen der Mariosoft haben sich dazu entschieden durch die Schwächen der Sprache und um die Anzahl der benutzten Programmiersprachen zu verringern, keine neuen Produkte mit PHP zu programmieren. Dadurch fehlt das nötige Personal mit PHP-Fachwissen. Außerdem wird das verwendete Framework nicht mehr unterstützt und das Update auf eine neuere Version bringt zu viele Breakingchanges mit sich.<sup>7</sup> Die Erweiterbarkeit und Anpassbarkeit ist bei diesem Rechner auch eingeschränkt, da das Frontend anhand von starren XML-Vorlagen, welche kompliziert zu erweitern sind, vorgegeben wird. Dadurch ist es schwierig Besonderheiten bei einigen Tarifen umzusetzen und diese zu warten. Die Oberfläche wird von den Kunden und Kundinnen als altmodisch wahrgenommen.

<sup>7</sup>s.[7] Zend-Framework

Hingegen läuft der OTR „Spirit“ auf einer neuer Technologie, ähnlich dem neuen OTR. Das Backend ist eine .NET 5.0 API und das Frontend eine Vue.js 2 SPA. Dieser Rechner wurde zwar mit einem generischem Ansatz geplant, allerdings lief die Logik schon beim Einführen des zweiten Tarifs zu weit auseinander. So wurde unter anderem eine andere Datenhaltungsmethode benutzt und die Struktur im Frontend war zu spezifisch zum ersten Tarif der umgesetzt wurde. Dadurch ist die Implementierung neuer Tarife nicht einheitlich und kompliziert.

Der neue OTR soll ein universeller Ansatz werden um alle zukünftigen Produkte gleich abzubilden. Durch die aktuellsten Technologien wird eine gewisse Langlebigkeit garantiert und durch den modularen und generischen Ansatz ist die Abbildung neuer Produkte einfacher. Durch ein Corporate-Design und die Einbindung der Marketing-Abteilung in der Konzeption der Benutzeroberfläche ist diese anschaulicher und benutzerfreundlicher.

### 3.3. Projektkosten

Die Kosten des Gesamtprojekts, also der kompletten OTR-Basis, belaufen sich auf ca. 80.000-100.000€. Dazu gehören hingegen mehrere Nebeneffekte die aus dem Gesamtprojekt resultieren. So wurde im Rahmen des OTRs die CI/CD Pipeline in GitLab neu aufgesetzt. Außerdem wurde ein einheitliches Corporate-Design eingeführt, an welchem die Marketingabteilung gearbeitet hat. Als erster Tarif in dem OTR wird eine Betriebshaftpflichtversicherung geplant, welche ausgearbeitet werden musste. Dazu kommt, dass das Bestandsführungssystem für die Betriebshaftpflichtversicherung konfiguriert werden muss.

Die Kosten der Fußzeilenkomponente lässt sich anhand der aufgewendeten Stunden berechnen. Dabei wird eine Arbeitsstunde eines Auszubildenden mit 100€ berechnet. Weitere benötigte Mitarbeiter, z.B. für Absprachen, lassen sich mit durchschnittlich 130€ pro Stunde berechnen. Dadurch kann folgende Kostenrechnung aufgestellt werden.

Für die Realisierung durch eines Auszubildenden:

$$69h * \frac{100\text{€}}{h} = 6900\text{€} \quad (1)$$

Tägliche Absprachen mit Entwicklungsteam:

$$1h * 4 \text{ Personen} * \frac{130\text{€}}{h} = 520\text{€} \quad (2)$$

Wöchentliche Absprachen bezüglich der Oberfläche:

$$1h * 6 \text{ Personen} * \frac{130\text{€}}{h} = 780\text{€} \quad (3)$$

Allgemeine regelmäßige Anforderungsabsprachen mit Auftraggeber:

$$5h * 1 \text{ Personen} * \frac{130\text{€}}{h} = 650\text{€} \quad (4)$$

Somit ergeben sind 8850€ an Kosten für die Fußzeilenkomponente.

$$6900\text{€} * 520\text{€} * 780\text{€} * 650\text{€} = 8850\text{€} \quad (5)$$

### 3.4. Amortisationsrechnung

Obwohl das Teilprojekt Funktionen beinhaltet, die für den Abschluss eines Versicherungsproduktes wichtig sind, kann der Nutzen nicht einzeln berechnet werden. Aufgrund der in 3.3 Projektkosten aufgeführten Gesamtkosten, müssten ca. 100.000€ erwirtschaftet werden damit sich der OTR rentiert. Dazu kommen allerdings viele Synergieeffekte, wie z.B. Zeitersparnisse bei der Entwicklung zukünftiger Versicherungsprodukte (s. 3.3 Projektkosten) die sich schwer berechnen lassen. Zeitgleich zum OTR entstand eine Betriebshaftpflichtversicherung, anhand welcher die Nutzen berechnet werden können.

Ausgehend davon, dass die K&M durch einen Vertrag 40€ pro Jahr verdient, müssten 2500 Verträge mit einer Haltedauer von einem Jahr abgeschlossen werden.

$$\frac{100.000\text{€}}{40\text{€} \frac{\text{Vertrag}}{\text{Jahr}}} = 2500 \text{ Verträge} \quad (6)$$

Demgegenüber kann davon ausgegangen werden, dass eine durchschnittliche Haltedauer von drei Jahren besteht. Damit sind 833 abgeschlossene Betriebshaftpflichtversicherungsverträge im ersten Jahr ausreichend um die Gesamtkosten des Projekts zu decken.

$$\frac{2500 \text{ Verträge}}{3 \text{ Jahre}} \approx 833 \text{ Verträge} * 3 \text{ Jahre} * 40\text{€} \approx 100.000\text{€} \quad (7)$$

Da die Betriebshaftpflichtversicherung ein neues Produkt der K&M ist, bestehen nicht genug Erfahrungswerte um genaue Angaben zu treffen. Anhand von bestehenden Versicherungsprodukten könnte dies allerdings eine realistische Amortisationsdauer sein.

## 4. Entwurfsphase

### 4.1. Zielplattform

Das gesamte Projekt soll in einem Container innerhalb eines Kubernetes Cluster laufen. Somit muss vorher ein Dockerimage gebaut werden um es dann in einem Container laufen zu lassen. Demnach ist die Zielplattform eine Linuxdistribution. Durch Kubernetes besteht ein Loadbalancing um eine bessere Skalierbarkeit zu ermöglichen. Ein Vorteil dieses Deployments ist zusätzlich, dass mehrere Versionen und unterschiedliche Stages die zeitgleich laufen vorhanden sind. Für jeden Feature-, Develop- und Masterbranch im Git wird jeweils eine eigene Instanz im Kubernetes Cluster erstellt, so dass jeder Stand auf dem Entwicklungssystem getestet werden kann. Dadurch ist die Anwendung besser testbar und neue Versionen können nahtlos veröffentlicht werden.<sup>8</sup>

### 4.2. Architekturdesign

Die Backend Architektur besteht aus Microservices nach dem Domain-Driven Design-Pattern. Außerdem wird für die Datenpersistenz das Repository-Pattern benutzt. Im Frontend werden Komponenten verwendet, welche ihre Daten über eine zentrale Datenzustandsverwaltung beziehen. Die Kommunikation zwischen Front- und Backend wird durch REST-API Aufrufe realisiert. Um die Daten synchron zu hal-

---

<sup>8</sup>s.[8] Using a GitLab CI/CD workflow for Kubernetes

ten wird jede Änderung im Frontend in den LocalStorage des Browsers geschrieben und zeitgleich per HTTP-JSON-PATCH Aufruf an das Backend weitergesendet. Das Backend speichert die neuen oder geänderten Daten im Cache und führt, wenn möglich, eine Berechnung sowie Validierung durch. Die berechneten Beiträge und Validierungsergebnisse werden anschließend zurückgesendet. Durch die generische JSON-PATCH Methode können Änderungen auch kleinteilig in Form von Schlüssel-Wert-Paaren bestehend aus dem Pfad in der JSON Struktur und dem Wert zwischen Front- und Backend synchronisiert und über das Backend persistiert werden. Das Persistieren über das Repository-Pattern wird eingehender in Abschnitt 4.4 Entwurf der Datenstruktur erklärt.

### 4.3. Entwurf der Benutzeroberfläche

Der designseitige Entwurf der Benutzeroberfläche wurde durch die Marketing-Abteilung per Mockups übernommen. Diese mussten allerdings nicht eins zu eins umgesetzt werden, sondern konnten in Kombination mit den Corporate-Design-Vorlagen als Richtlinie genutzt werden. Weiterhin gab es wöchentliche Absprachen in denen speziell das Design abgestimmt wurde. Dadurch musste weniger Zeit für die Entwicklung und Entwerfen der Oberfläche und Workflows von meiner Seite aus aufgebracht werden. Abbildungen eines Mockups, das finale Design der Fußzeilenkomponente und die Fußzeile im Gesamtkontext des OTR sind jeweils im Anhang zu finden (s. A.2 Mockup-Design der Fußzeile, A.2 Tatsächliches Design der Fußzeile, A.2 Benutzeroberfläche OTR)

### 4.4. Entwurf der Datenstruktur

Wie schon in 4.2 Architekturdesign erwähnt, wird das Repository-Pattern angewendet um Daten zu persistieren. Dementsprechend gibt es verschiedene Repository-Klassen, welche den Datenzugriff kapseln. Der Datenzugriff ist hierbei zweistufig. Als erstes werden gespeicherte Angebote, Tarifkonfiguration oder andere übergebene Daten im Cache gespeichert. Dieser ist nur ein temporärer Speicher der die Daten für bis zu sechs Stunden im Arbeitsspeicher hält. Die zweite Stufe ist der Distributed Cache (Storage). Der Storage speichert die Daten in einer Datenbank ab. Wenn, wie in 1.4 Projektschnittstellen dargestellt, Angebote über das Maklerportal gespeichert werden, werden die Daten fünf Jahre lang in der Datenbank persistiert. Eine andere Möglichkeit um Daten vom Cache in den Storage zu überführen ist der Aufruf der Shutdown-Action im Backend. Dabei werden die Daten einen Tag lang gespeichert. Benutzt wird Shutdown z.B. wenn der OTR neugestartet werden muss und die Daten dementsprechend aus dem Cache kurzzeitig gespeichert werden müssen damit diese nicht verloren gehen.

Bei einem Datenzugriff wird immer als erstes der Cache abgefragt und wenn die geforderten Daten dort nicht zu finden sind, der Storage.<sup>910</sup>

Für die Zustandsverwaltung der Daten im Frontend wird der Pinia-Store benutzt. Dieser ist typischer, wodurch das Arbeiten mit Typescript erleichtert wird. Allgemein können mit so einem Store die Daten komponentenübergreifend einheitlich gespeichert werden. In dem Frontend des OTR werden in dem Pinia-Store wichtige Daten, welche das Backend bei der Initialisierung liefert, gespeichert. Dazu gehören unter anderem Konfigurationen, wie die Oberfläche des OTRs aufgebaut werden soll, welche Eingabemöglichkeiten gege-

---

<sup>9</sup>s.[9] Cache in-memory in ASP.NET Core

<sup>10</sup>s.[10] Verteilter Cache

ben sind und eventuelle Sitzungsdaten, wenn eine bestehende Sitzung geladen wurde. Außerdem werden berechnete Beiträge sowie eingegebene Daten gespeichert. Der Zugriff auf die Daten aus anderen Komponenten erfolgt über Gettermethoden, welche im Store definiert werden.

#### **4.5. Entwurf der Geschäftslogik**

Die Logik der Benutzeroberfläche und die technische Umsetzung wurden innerhalb des Entwicklungsteams durch Prototyping erarbeitet.<sup>11</sup> Wie schon in 1.2 Projektziel beschrieben, wird die Oberfläche unter anderem mit Vue.js realisiert. Der große Vorteil von Vue.js ist, dass die Oberfläche reaktiv gemacht wird. Somit ändert sich das Oberflächenelement automatisch, wenn sich die zugehörigen Daten ändern. Dies geschieht über das Observer-Designpattern.

<sup>12</sup> Für den OTR wichtige Daten werden in dem Store gespeichert. Die Fußzeilenkomponente greift hierbei auf die Konfiguration des OTRs sowie die Sessionguid zu. Eine weitere Funktion des Stores ist ein Benachrichtigungssystem. Hierbei wird bei jeder Änderung von bestimmten Daten ein Ereignis ausgelöst, so dass eine Benachrichtigung, z.B. ein Fehler oder eine Erfolgsnachricht bei einer Aktion, ausgegeben wird.

Die Geschäftslogik bzw. das Backend ist eine .NET API. Als Programmiersprache wird dementsprechend C# 10 benutzt. Im Backend werden Datenhalterklassen anhand von XML Schemata automatisch generiert. Mit Hilfe von der Swagger OpenApi Dokumentation der Controller wird ein TypeScript API-Client geschaffen. Genauso werden C#-Clients für benutzte Schnittstellen, z.B. dem Dokumentenversandservice, erzeugt. Für die einzelnen Controller der API bestehen Serviceklassen in welchen die Funktionalität bereitgestellt wird. So gibt es ein Interface welches die benötigten Methoden vorgibt und eine abstrakte Klasse welche alle tarifunabhängigen Methoden des Interfaces implementiert. Zudem gibt es für jeden Tarif oder jedes Versicherungsprodukt eine Klasse, welche von der abstrakten Klasse erbt und das Interface implementiert. Dadurch kann im Controller dynamisch anhand vom jeweilig aktivem Tarif die passende Implementierung benutzt werden. Alle Abhängigkeiten werden per Dependency Injection aufgelöst.

#### **4.6. Maßnahmen zur Qualitätssicherung**

Durch die Arbeit mit GitFlow wurde jede Unteraufgabe in einem eigenem Gitbranch bearbeitet. Diese Branches werden erst nach Codereviews vom Entwicklungsteam in den *develop*-Branch überführt. Bei jedem Push werden per GitLab Pipeline Unittests im Front- und Backend ausgeführt. Im Backend werden die bestehenden Unittests erweitert um neu implementierte Logik zu testen. Zudem wurde im Frontend ein Lint konfiguriert um einheitliche Codeguidelines und Codestyling durchzusetzen.

Wenn eine Änderung auf den Developbranch gepusht wurde wird diese von einem anderen Entwickler oder Entwicklerin getestet. Außerdem gibt es weitere Mitarbeiter/-innen außerhalb des Entwicklerteams die regelmäßig auf dem Testsystem Blackbox-Akzeptanztests durchführen.

---

<sup>11</sup>s.[11] Software Prototyping, Wikipedia

<sup>12</sup>Vgl.[12] Reactivity in Depth

## 5. Implementierungsphase

### 5.1. Implementierung der Datenstruktur

Die Datenstruktur wie in 4.4 Entwurf der Datenstruktur beschrieben, war schon gegeben. Dementsprechend konnte der Pinia-Store im Frontend sowie die Speicherung über die Repositorys im Backend für die Implementierung der Fußzeile genutzt werden.

Für den Store mussten einzelne Abfragen in Form von Gettermethoden implementiert werden, wie z.B. die Überprüfung ob ein Dokument druckbar ist. Die nötigen Funktionalitäten werden in 5.2 Implementierung der Benutzeroberfläche und Frontendlogik weiter aufgezeigt.

### 5.2. Implementierung der Benutzeroberfläche und Frontendlogik

Als Fußzeilencontainer war die `<footer>` Komponente von Vuetify gegeben. Die einzelnen Schaltflächen wurden mit `<km-button>` umgesetzt. Dieser ist in der besagten K&M Komponentenbibliothek gegeben. Eine Besonderheit ist hierbei die Schaltfläche für den Dokumentenversand. Da der Dokumentenversand relativ viel Eigenlogik benötigt und schon in einer vorherigen OTR Generation vorhanden war, gibt es diesen als eigenes npm-Paket. Im Endeffekt wird in dieser Komponente allerdings auch nur der `<km-button>` verwendet. Die Einbindung des Pakets gestaltete sich aufgrund fehlender Dokumentation anspruchsvoller als angenommen.

Die Schaltfläche zur Speicherung von Angebot und Kopie soll nur vorhanden sein, wenn der OTR über das Maklerportal aufgerufen wurde. Für so etwas bietet Vue.js ein konditionales Rendering als HTML-Attribut an. Im Pinia-Store wird überprüft, ob das Angebot speicherbar sein soll und dieser Wert wird an das Attribut übergeben. Beim Drücken auf die Schaltfläche wird ein JavaScript-Fetch Aufruf ans Backend gemacht. Anhand der Schnittstellenbeschreibung durch NSwag wurde ein Client als Typescriptklasse generiert. Dieser kümmert sich um die Fehlerbehandlung anhand des HTTP-Statuscodes, erwartet die Argumente als richtigen Datentyp und gibt den Response als eigenen Datentyp zurück. Ein Angebot wird anhand einer Globally Unique Identifier (GUID) gespeichert, welche die aktuelle Sitzung des OTRs repräsentiert. Außerdem wird die Identifikationsnummer des angemeldeten Maklers oder der Maklerin bzw. Vertriebspartners/Vertriebspartnerin benötigt. Dementsprechend werden diese Daten mit an das Backend übergeben damit dieses sich anhand der Session die Angebotsdaten aus dem Cache laden und diese dann an das TRG zum Speichern weiterschicken kann. Auf der Seite des Frontends geschieht bei der Kopiespeicherung das Gleiche wie bei der normalen Speicherung, außer dass ein anderer Endpunkt aufgerufen wird. Die Logik im Backend unterscheidet sich dabei von der Angebotsspeicherung.

Die Angebot- und Antragsdrucken Schaltflächen sind zwar immer vorhanden, sind allerdings nicht immer aktiviert bzw. benutzbar. Diese Logik setzt sich aus zwei Bedingungen zusammen. Zum einem muss überhaupt für das aktuelle Produkt der Antrags- bzw. Angebotsdruck aktiviert sein. Zum anderen ist das Drucken erst möglich, wenn ein Beitrag berechnet werden konnte. Die notwendigen Daten werden aus dem Pinia-Store geladen, überprüft und bestimmen in den Schaltflächenkomponenten ob sie aktiviert sind oder nicht. Beim Drücken der Schaltflächen wird ein Backend Aufruf an die Drucklogik gestartet. Dieser erwartet die Ses-

sionguid, damit sich das Backend die benötigten Daten aus dem Speicher laden kann. Zurückgeliefert wird ein PDF-Dokument als Filestream. Das wird dann im Frontend aufbereitet, so dass automatisch ein PDF-Dokument heruntergeladen wird.

In die Fußzeile kommt dazu noch das Impressum und das Copyright. Das Impressum ist hierbei eine Verlinkung auf das Impressum der Homepage von K&M.

### **5.3. Implementierung der Geschäftslogik**

Das Backend stellt mehrere Services bereit, gemäß Domain-Driven Design. Diese sind in verschiedene Projekte aufgeteilt. Unter anderem gibt es im Backend Schnittstellen für Anträge, Beitragsberechnung, Tarifkonfigurationen und allgemeine Logik. Jede dieser Schnittstellen hat nach dem Domain-Driven Design ein eigenes Projekt für die eigentliche Schnittstelle, für die Infrastruktur und für die Logik. Zusätzlich wird die Logik für die verschiedenen Versicherungstarife aufgeteilt, da die Produkte oder Versicherungstarife verschieden sind.<sup>13</sup> Die unterschiedliche Logik der Tarife wird dynamisch je nach aktivem Tarif benutzt. Das wird durch Dependency Injection ermöglicht. Für die Fußzeilenkomponente ist nur der Management-service interessant, da dort allgemeingültige und tarifunspezifische Logik zu finden ist.

Zum Speichern und Kopieren von Angeboten wird die Session als GUID benötigt. Anhand dieser Sessionguid, werden die Konfiguration und die gespeicherten Angebotsdaten aus dem Cache geladen. Sowohl beim Speichern als auch beim Kopieren wird größtenteils die gleiche Logik benutzt. So wird bei beiden Varianten die Sessionspeicherungsschnittstelle des TRGs aufgerufen, welches einen Eintrag in einer Datenbank hinterlegt oder falls schon einer mit dieser Sessionguid vorhanden ist, bearbeitet. Anschließend wird das Angebot im Storage gespeichert.

Die Besonderheit bei der Kopie ist, dass vor dem Aufruf ans TRG eine neue GUID erzeugt wird. Dadurch wird ein neuer Eintrag in die Datenbank geschrieben. Außerdem muss das Angebot mit der neuen Sessionguid auch im Cache gespeichert werden. Die neue Sessionguid wird nun ans Frontend zurückgegeben, damit dort mit der Kopie weitergearbeitet wird.

Bei den Aufrufen für das Drucken von einem Angebot oder Antrag wird die Sessionguid und der Produkttyp, also welcher Tarif, benötigt. Hierbei wird der Pdftoolsservice aufgerufen, eine interne Schnittstelle die PDF-Dateien anhand eines Angebots bzw. Antrags als Stream liefert. Vorher muss allerdings, genau wie im Frontend, überprüft werden ob ein PDF überhaupt druckbar ist. Dafür werden die eingegebenen Daten validiert und eine Beitragsberechnung durchgeführt. Dieser Vorgang wird, obwohl diese Bedingungen im Frontend geprüft werden, nochmal Backendseitig vollzogen. Da Angebote und Anträge rechtlich bindend sind muss sichergestellt werden, dass diese Bedingungen erfüllt sind. Erst wenn die Überprüfung erfolgreich ist, wird ein PDF zurückgeliefert. Da diese Logik, bis auf den Aufruf vom Pdfservice identisch ist, wird der Pdfserviceaufruf per anonyme Funktion als Argument übergeben.

Die PDF-Dateien des Angebots und Antrags können zusätzlich per E-Mail versendet werden und aus diesem Grund benutzt die Dokumente-Versenden-Logik den gleichen Programmcode wie das Drucken. Wiederum können auch andere Dokumente versendet werden. Diese müssen beim Aufruf der Routine mit angegeben werden. Die Dokumente werden dann heruntergeladen und anschließend wird ein Aufruf an den Mailservice

---

<sup>13</sup>Vgl.[13] Design a DDD-oriented microservice

getätigt, welcher diese Dokumente an die angegebene E-Mail versendet.

## 6. Projektabschluss

### 6.1. Abnahme

Durch die regelmäßigen Absprachen und Codereviews durch das Entwicklungsteam sowie die wöchentlichen Absprachen bezüglich der Oberfläche gab es keine formale Abnahme der Fußzeilenkomponente. Alle anfallenden Anforderungen konnten im Laufe der Implementierung umgesetzt werden. Ausgenommen davon sind Anforderungen die kurz vor Abschluss des Projekts aufgetaucht sind. Auf diese wird in 6.4 weiter eingegangen. Außerdem wurde regelmäßig die Funktionalität getestet, so dass jeder Bestandteil einzeln getestet und abgenommen wurde.

### 6.2. Soll-Ist-Vergleich

In dem Soll-Ist-Vergleich wird gezeigt, wie schon in 4.3 Entwurf der Benutzeroberfläche erwähnt, dass durch die bestehenden Mockups und Absprachen weniger Zeit bei der Entwicklung und dem Entwerfen der Oberfläche benötigt wurde, als geplant. Diese gewonnene Zeit ist allerdings in die wöchentlichen Absprachen zu der Oberfläche, die Analyse bestehender Codebasen sowie Implementierung der Frontendlogik geflossen.

Projektphasen	Soll in Std	Ist in Std	Differenz
Tägliche Absprachen mit Entwicklungsteam & Auftragsgeber	1	1	0
Wöchentliche Absprache zur Oberfläche	0	1	+1
<b>Ist-Analyse:</b>	<b>5</b>	<b>6</b>	<b>+1</b>
Analyse vorhandener Codebasen und Funktionalitäten	2	3	+1
Gespräche mit dem Auftragsgeber	3	3	0
<b>Sollkonzept:</b>	<b>5</b>	<b>4</b>	<b>-1</b>
Planung des Soll-Zustands Projekt allgemein	2	2	0
Planung des Soll-Zustands Fußzeilenfunktionalität	1	1	0
Abstimmung & Sichtung Workflows und Oberflächenentwürfe	2	1	-1
<b>Realisierung:</b>	<b>31</b>	<b>30</b>	<b>-1</b>
<b>Frontend</b>	<b>10</b>	<b>12</b>	<b>+2</b>
Entwicklung Oberfläche	4	1	-3
<b>Backend</b>	<b>17</b>	<b>17</b>	<b>0</b>
<b>Dokumentation</b>	<b>19</b>	<b>19</b>	<b>0</b>
<b>Testen &amp; Abnahme</b>	<b>8</b>	<b>8</b>	<b>0</b>
<b>Gesamt</b>	<b>69</b>	<b>69</b>	<b>0</b>

Tabelle 3: Vergleich gebrauchte Zeit der Projektphasen

Von dem geplanten Umfang des Teilprojekts wurde alles erfüllt. Zudem wurde die in 2.2 Abweichung vom



Projektantrag erwähnte Angebotskopiespeicherung implementiert.

### **6.3. Gewonnene Erkenntnisse**

Die Umsetzung dieses Projekts brachte wertvolle Erkenntnisse. Es wurde das erste Mal mit Vue.js 3 entwickelt, was im Gegensatz zum Vorgänger einige Änderungen mit sich brachte. Dadurch wurde im Bereich der Frontendentwicklung viel neues gelernt. Vuetify bzw. die Komponentenbibliothek half bei einem einheitlichem Design und erleichterte die Einbindung durch gestellte Vorlagen. Die durch TypeScript bereit gestellten Typsicherheit machte das Arbeiten im Team einfacher. Das Benutzen von Bibliotheken, vorausgesetzt diese haben Typescriptdatentypen deklariert, wurde auch erleichtert. Außerdem wird sich die Wartung und die Umsetzung zukünftiger Erweiterungen problemloser gestalten.

Dazu kommt, dass das erste Mal mit gegebenen Oberflächenmockups gearbeitet wurde, wodurch das Designen und Entwerfen der Oberfläche erleichtert wurde. Weiterhin ist durch die Konzeption des Gesamtprojekts von einem Softwarearchitekten eine einheitliche Struktur erkennbar, was das Implementieren neuer Logik und das Anpassen oder Erweitern bestehender Logik einfacher gestaltete.

### **6.4. Ausblick**

Als neuste Generation der OTR sollen nun alle neuen Tarife anhand dieser Basis implementiert werden. Momentan ist geplant mindestens einen alten Tarif in die neue Basis zu übertragen.

Die Konfigurationen für alle Tarife sind zur Zeit in verschiedenen Systemen verstreut. Geplant ist es diese in einer NoSQL Datenbank mit Anbindung zum neuen OTR zu speichern, so dass der OTR die zentrale Stelle für Tarifkonfigurationen wird.

Für die Fußzeilenkomponente sind noch kleinere Erweiterungen ausstehend, die nicht im Rahmen des Abschlussprojektes umgesetzt werden konnten oder nicht geplant waren. Dazu gehört eine Ladeanimation und Eingabesperre im Frontend, solange auf das Drucken eines Dokuments gewartet wird. Hinzufügend sind kleinere Refactorings im Backend sowie ein Refactoring der Datenzustandsverwaltung im Frontend geplant.

## Literaturverzeichnis

- [1] “Vue.js.” <https://vuejs.org/>, 2022. Zugriff am 10.03.2022.
- [2] “Vuetify.” <https://github.com/vuetifyjs/vuetify>, 2022. Zugriff am 07.03.2022.
- [3] “Corporate Design.” [https://de.wikipedia.org/wiki/Corporate\\_Design](https://de.wikipedia.org/wiki/Corporate_Design), Wikipedia, 2021. Zugriff am 06.03.2022.
- [4] “Extensible Markup Language.” [https://de.wikipedia.org/wiki/Extensible\\_Markup\\_Language](https://de.wikipedia.org/wiki/Extensible_Markup_Language), Wikipedia, 2022. Zugriff am 16.03.2022.
- [5] “What is Pinia?.” <https://pinia.vuejs.org/introduction.html>, 2022. Zugriff am 14.03.2022.
- [6] “Spiralmodell: Das Modell für Risikomanagement in der Softwareentwicklung.” <https://www.ionos.de/startupguide/produktivitaet/spiralmodell/>, 2019. Zugriff am 16.03.2022.
- [7] “Zend-Framework.” <https://framework.zend.com/manual/1.12/en/introduction.overview.html>, ? Zugriff am 14.03.2022.
- [8] “Using a GitLab CI/CD workflow for Kubernetes.” [https://docs.gitlab.com/ee/user/clusters/agent/ci\\_cd\\_workflow.html](https://docs.gitlab.com/ee/user/clusters/agent/ci_cd_workflow.html), 2021. Zugriff am 16.03.2022.
- [9] “Cache in-memory in ASP.NET Core.” <https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-6.0>, 2022. Zugriff am 13.03.2022.
- [10] “Verteilter Cache.” [https://de.wikipedia.org/wiki/Verteilter\\_Cache](https://de.wikipedia.org/wiki/Verteilter_Cache), Wikipedia, 2021. Zugriff am 10.03.2022.
- [11] “Software Prototyping.” [https://en.wikipedia.org/wiki/Software\\_prototyping](https://en.wikipedia.org/wiki/Software_prototyping), 2022. Zugriff am 11.03.2022.
- [12] “Reactivity in Depth.” <https://vuejs.org/guide/extras/reactivity-in-depth.html>, 2022. Zugriff am 11.03.2022.
- [13] “Design a DDD-oriented microservice.” <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice>, 2022. Zugriff am 14.03.2022.

## **Abkürzungsverzeichnis**

**K&M** Konzept und Marketing GmbH  
**SPA** Single-Page-Application  
**OTR** Onlinetarifrechner  
**GUID** Globally Unique Identifier  
**TG** Tarifrechner-Gateway  
**TRG** Tarifrechner-REST-Gateway  
**CI/CD** Continuous Integration/Continuous Delivery

## Glossar

**.NET** Eine Softwareplattform für das Entwickeln und Ausführen von Anwendungen. 1, 5, 6, 10

**API** Application Programming Interface, Programmierschnittstelle; ein Programmteil der zur Anbindung für andere Systeme zur Verfügung gestellt wird. 10

**Container** Laufende Instanz eines Softwarepakets im Dockerumfeld. 8

**Corporate-Design** Einheitliches Erscheinungsbild eines Unternehmens. 1, 5, 7, 9

**Dependency Injection** Ein Entwurfsmuster welches Abhängigkeiten von Objekten zur Laufzeit auflöst. 10, 11

**Distributed Cache** Verteilter Cache, Speicher der sich über mehrere Instanzen der Anwendung spannt. 9

**Domain-Driven Design** Modellierung der Software nach umzusetzenden Fachlichkeiten der Anwendungsdomäne. 8, 11

**Git** Dezentrale Versionverwaltung. 1

**GitFlow** Ein Modell für das Benutzen von Gitbranches, in welchem eigenständige Anforderungen in eigene Branches aufgeteilt werden. 1

**GitLab** Ein Anbieter von Git mit DevOps-Funktionen. 1, 7

**HTML** Hypertext Markup Language; textbasierte Auszeichnungssprache um Inhalte in Webbrowsern darzustellen. 11

**HTTP** Hypertext Transfer Protocol; Protokoll für Datenübertragung. 11

**Javascript-fetch** Asynchroner Aufruf an einen Endpunkt. 11

**Kubernetes** Orchestrator für Containeranwendungen. 1, 8

**Linter** Ein Tool zur statischen Codeanalyse. 10

**LocalStorage** Ein persistenter, clientseitiger Datenspeicher im Webbrowser. 3, 8

**npm** Ein Paketmanager für Node.js Pakete. 10

**NSwag** Ein Tool, um Swagger OpenApi Beschreibungen und Klassen aus dieser Beschreibung zu generieren. 11

**PHP** Eine Skriptsprache die zur Erstellung dynamischer Webseiten dient. 5, 6

**Pinia-Store** Eine Datenzustandsverwaltung für Vue.js, welche den Localstorage des Browsers verwendet. 3, 9, 11

**Repository-Pattern** Ein Designpattern, welches die Logik der Datenspeicherung und -beschaffung kapselt, so dass eine zentrale Stelle für Datenquellen besteht. 8, 9

**TypeScript** Eine typsichere Erweiterung von JavaScript. 1, 10, 13

**Vue.js** Ein clientseitiges Javascript-Framework nach dem Model-View-ViewModel-Entwurfsmuster für das Erstellen von Webanwendungen. 1, 3, 5, 6, 9, 11, 13

**Vuetify** Eine Vue.js Oberflächenkomponentenbibliothek. 1, 10, 13

**XML** eXtensible Markup Language; eine Auszeichnungssprache zur Abbildungen strukturierter Daten. 2, 6, 10

## A. Anhänge

### A.1. Anwendungsfalldiagramm

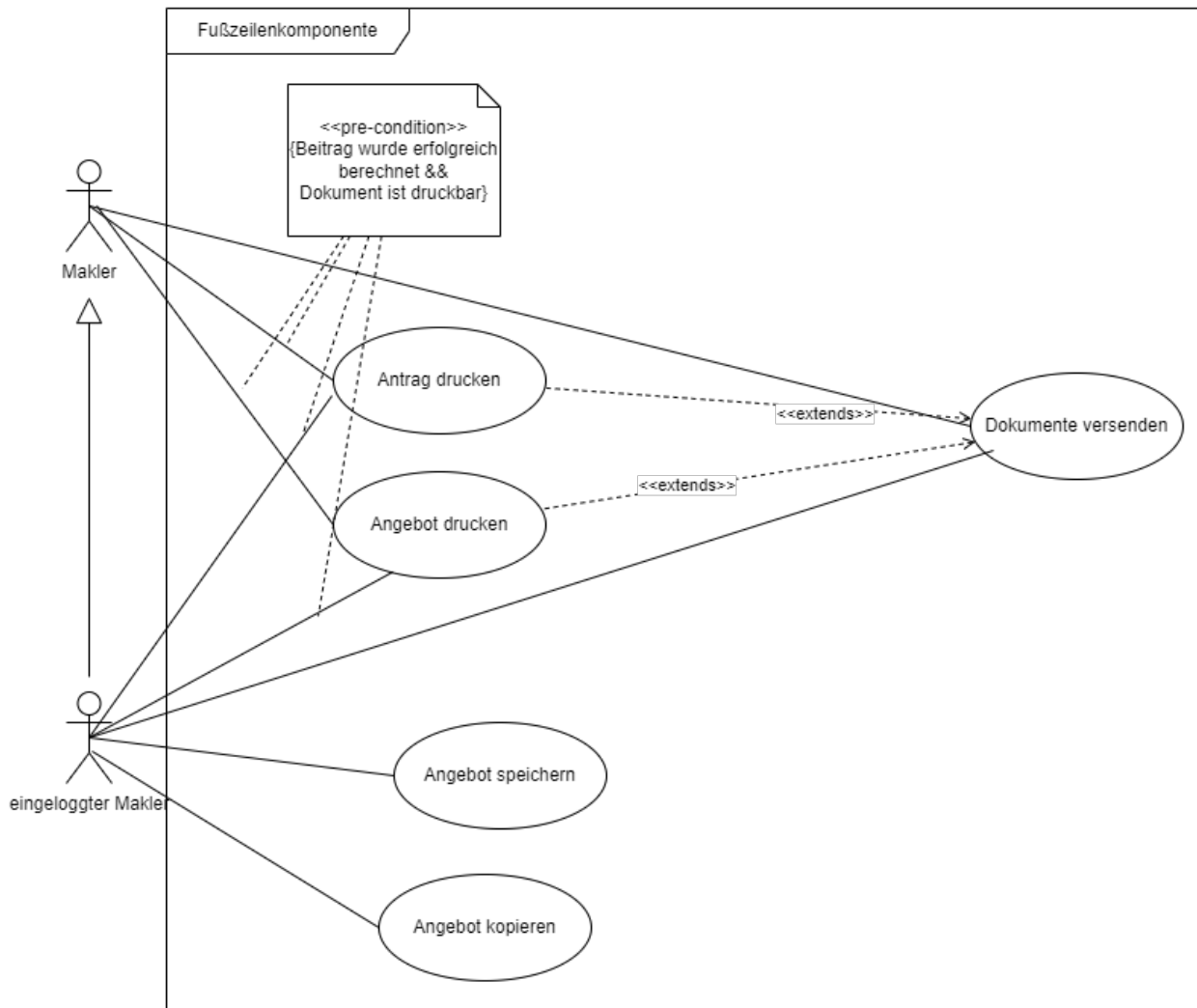


Abbildung 1: Anwendungsfalldiagramm der Fußzeilenkomponente

### A.2. Oberflächenabbildungen

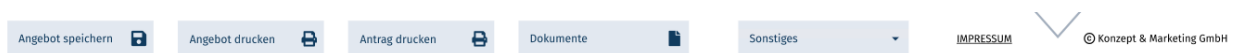


Abbildung 2: Mockup-Design der Fußzeile



Abbildung 3: Tatsächliches Design der Fußzeile

## K&M Betriebshaftpflicht

Tarif 1

Bausteine

Annahmefragen 4

Sonstige Angaben 4

Versicherungsnehmer 7

Übersicht

Abschluss 7

23/27 Angaben ausstehend

Tarif

Betriebsart

Betriebsart

Im Betrieb tätige Personen

Im Betrieb tätige Personen  
1

Auswahl der Versicherungssumme

Auswahl der Versicherungssumme  
3 Mio. €

Tarifstärken

Betriebshaftpflichtversicherung

Einfach zuverlässig

✓ Keinerlei Selbstbehalte

✓ Tätigkeitsschäden auch anlässlich von Reparaturen im Betrieb


✓ Verlust fremder Schlüssel bis VSU, Folgeschäden bis 25.000 Euro

✓ Filial- und Zweigniederlassungen, Verkaufsstellen und Lagerstätten sind mitversichert

✓ Schäden an gemieteten Gebäuden und Räumen

✓ Keine Ratenzahlungszuschläge

Bruttojahresbeitrag 0,00 €



Ihr Vertrag

Versicherungsumfang

Betriebsart:

Anzahl Personen: 1

Versicherungssumme: 3 Mio. €

Grunddeckung: 0,00 €

Bausteine

Leistungsgarantie: 0,00 €

Zahlweise

Zahlweise  
monatlich

Courtageart

Courtage in Proz... 25

Courtagesatz 0,00 %

Courtagebetrag 0,00 €

Nettozahlbeitrag 0,00 €

Versicherungsteuer (0%) 0,00 €

Bruttozahlbeitrag 0,00 €

← VORHERIGER SCHRITT

NÄCHSTER SCHRITT →

Angebot speichern

Angebot kopieren

Angebot drucken

Antrag drucken

Dokumente

IMPRESSUM

© Konzept & Marketing GmbH

Abbildung 4: Benutzeroberfläche OTR

Jakob Schumann

V