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 3

Prüfungsbewerber:
Prüflingsnummer: 94023
Jakob Schumann
Goethestraße 40
30169 Hannover

Ausbildungsbetrieb:
Konzept & Marketing – ihr unabhängiger Konzeptentwickler GmbH
Podbielskistraße 333
30659 Hannover

Inhaltsverzeichnis

Abbildungsverzeichnis

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. 11

API Application Programming Interface, Programmierschnittstelle; ein Programmteil der zur Anbindung für andere Systeme zur Verfügung gestellt wird. 11

Container Laufende Instanz eines Softwarepakets im Dockerumfeld. 9

Dependency Injection Ein Entwurfsmuster welches Abhängigkeiten von Objekten zur Laufzeit auflöst. 11, 13

Distributed Cache Verteilter Cache, Speicher der sich über mehrere Instanzen der Anwendung spannt. 10

Domain-Driven Design Modellierung der Software nach umzusetzenden Fachlichkeiten der Anwendungsdomäne. 9, 13

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. 12

HTTP Hypertext Transfer Protocol; Protokoll für Datenübertragung. 12

Javascript-fetch Asynchroner Aufruf an einen Endpunkt. 12

Kubernetes Orchestrator für Containeranwendungen. 1, 9

Linter Ein Tool zur statischen Codeanalyse. 11

Localstorage Ein persistenter clientseitiger Datenspeicher im Webbrowser. 3, 9

npm Ein Paketmanager für Node.js Pakete. 10, 12

NSwag Ein Tool um Swagger OpenApi Beschreibungen und Klassen aus dieser Beschreibung zu generieren. 12

Pinia-Store Eine Datenzustandsverwaltung für Vue.js, welche den Localstorage des Browsers verwendet. 10, 12, 13

Repository-Pattern Ein Designpattern, welches die Logik der Datenspeicherung und -beschaffung kapselt, so dass eine zentrale Stelle für Datenquellen besteht. 9, 10

TypeScript Eine typsichere Erweiterung von JavaScript. 1, 10, 11

Vue.js Ein clientseitiges Javascript-Framework nach dem Model-View-ViewModel-Entwurfsmuster für das Erstellen von Webanwendungen. 1, 10, 12

Vuetify Eine Vue.js Oberflächenkomponentenbibliothek. 1, 10–12

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 über freie Makler über die Vertragsannahme, bis zur Schadenregulierung. Das Unternehmen, bestehend derzeit aus 118 Mitarbeitern, bietet Dienstleistungen sowie Portale für Versicherer, Makler und Kunden an.

Mariosoft

Bei der Mariosoft handelt es sich um eine IT-Abteilung der K&M. Die Mariosoft besteht derzeit aus 18 Mitarbeitern 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 bestehen aus einem .NET 6.0 Backend, welches Schnittstellen zu umliegenden firmeninternen Systemen hat. Außerdem ist ein Cachespeicher und weitere Logik vorhanden. Das Frontend besteht aus Vue.js mit Designvorlagen von Vuetify sowie einer eigenen Oberflächenkomponentenbibliothek, welche Vuetifyelemente kapselt und im Corporatedesign zur Verfügung stellt. Im Frontend wird TypeScript als Programmiersprache benutzt.
Als Versionsverwaltung für dieses Projekt wird Git mit GitFlow genutzt. Das Continuous Integration/Continuous Delivery (CI/CD) wird mittels GitLab Pipelines realisiert, also 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 wird auf das Verfahren weiter eingegangen.
Bei einem OTR hat ein Makler die Möglichkeit Beiträge für versicherte Risiken zu errechnen. Außerdem 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 von mir im Rahmen dieses Projekts umgesetzt. Die Fußzeile beinhaltet außerdem Angaben zum Copyright und einen

Verweis zum Impressum.

Beim Speichern eines Angebots sollen alle im OTR eingegebenen Daten im sogenannten Maklerportal für den angemeldeten Makler gespeichert werden. Diese können über das Maklerportal wieder geladen werden. Außerdem ist es vorgesehen, dass Makler Angebote kopieren können. Die K&M bietet auch OTR außerhalb des Maklerportals an. Für diese Rechner ist dann keine Angebots- oder Kopiespeicherung vorgesehen, dementsprechend sind die Schaltflächen dort nicht vorhanden.
Die Druckenfunktionalität wird durch zwei Schaltflächen abgebildet, eine um ein Angebot zu drucken, die zweite für den Antrag. Ein Angebot wird z.B. dem Endkunden vom Makler unterbreitet und ist, genau 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 Angebot- 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 es sein, dass für einen Tarif der Angebots- oder Antragsdruck deaktiviert ist. Dementsprechend sind die Schaltflächen aktiviert oder nicht aktiviert. Das Drucken ist dabei eigentlich das Herunterladen eines PDF-Dokumentes. Das Dokument wird von einem angebundenem System anhand der Daten in XML-Format generiert.
Eine weitere Funktion ist die des Dokumentenversands. Bei Drücken dieser Schaltfläche öffnet sich ein Dialog, wo der Nutzer die gewünschten Dokumente aus allen relevanten Dokumenten auswählen kann um sie zu versenden. Dazu gehören auch die zuvor beschriebenen Angebots- und Antragsdokumente. Dazu muss noch eine oder mehrere E-Mailadressen angegeben werden und dann werden die Dokumente an diese E-Mailadressen in einem ZIP-Archiv versendet. Die Dokumente beinhalten rechtliche Informationen und den Antrag sowie das Angebot. Die komplette Dokumentenversand-Komponente gibt es als eigenständige Bibliothek, 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. Der OTR ist 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 die Fußzeile bietet sich dafür gut an, da sie unabhängig von der übrigen Webseite funktionieren soll.

1.4. Projektschnittstellen

Das Backend hat Schnittstellen zu einigen Systemen. Es gibt 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) geliefert.
Das Maklerportal ist eine Webseite an der sich Makler 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 auch gespeicherte Angebote eingesehen, bearbeitet und mit einem OTR aufgerufen werden.
Der OTR wird vor allem von Maklern benutzt um Tarife für ihre Kunden 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. Genauso ist eine Synchronisation zwischen Front- und Backend durch HTTP-JSON-PATCH schon vorhanden. Weiterhin ist ein Backend mit mehreren Schnittstellen welche nur für die einzelnen Funktionalitäten erweitert werden müssen verfügbar. Im Frontend ist ein Gridlayout mit Elementvorlagen gegeben, welche genauso erweitert werden müssen.

2. Projektplanung

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

2.1. Projektphasen

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

2.2. Abweichung vom Projektantrag

Alle im Projektantrag beschriebenen Funktionalitäten wurden umgesetzt. Hinzu kam die Kopiespeicherung eines Angebots. Außerdem war schon ein Grundgerüst in Form von bestehenden Serviceprojekten und Schnittstellen gegeben. Die UI-Elemente mussten teilweise neu erstellt, erweitert oder angepasst werden. Dadurch, dass Mockups für die Benutzeroberfläche gegeben sind, musste weniger Zeit von meiner Seite aus in die Konzeption der Benutzeroberfläche fließen. Deswegen weichen die angegebenen Zeiten aus Abschnitt 2.1 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 wurde mit dem Spiralmodell durchgeführt, unterstützt durch ein Kanban-Board. Vor dem Start der Bearbeitung wurden grobe Aufgaben und Anforderungen definiert. Es gab tägliche Absprachen zwischen dem Entwicklerteam sowie wöchentliche Absprachen mit dem Produktmanagement und der Marketingabteilung bezüglich der Oberfläche. Aus diesen Absprachen sind iterativ neue Anforderungen entstanden. Durch CI/CD wurden inkrementell umgesetzte Funktionalitäten bzw. behobene Fehler zum Testen zur Verfügung gestellt.
Dieses Vorgehensmodell wurde gewählt damit man die Entwicklung schnell an neue Anforderungen der Kunden bzw. der Stakeholder anpassen kann.

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 aufgebaut mit einem .NET-Backend und einem Vue.js Frontend. 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 den Entwickler schwierig zu warten und zu erweitern.

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 den Endkunden 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 an. Dieses Versicherungskonzept könnte bei marktüblichen Baukastenrechnern nur über mehrere Rechner kalkuliert werden.
Da es schon ältere Generationen an OTR gibt, wäre auch eine Überlegung einfach eine ältere Onlinetarifrechnerbasis weiter zu benutzen anstatt einen neuen einzuführen. Die älteste Generation ist allerdings technologisch nicht tragbar und muss grundlegend erneuert werden. Der neuste OTR sollte die Grundlage werden, allerdings 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 wurden 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 den Endkunden wichtig ist. Die Bewertung geht von 0-10 Punkten, wobei 10 die best möglichste Bewertung ist.

		1. Generation OTR ('Universal')		2. Generation OTR ('Spirit')		3. Generation OTR	
Kriterium	Gewichtung	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
Summe	100%	11	**27,5**	25	**61**	33	**81**

Der Rechner 'Universal' ist in PHP programmiert. Die Entwickler 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. 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 umzuset-

zen und diese zu warten. Die Oberfläche ist sehr altmodisch.
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 es einfach neue Produkte abzubilden. 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 allerdings 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. Demnach ergibt sich daraus:
Für die Realisierung durch Auszubildenden:
69 Std * 100€/Std = 6900€
Tägliche Absprachen mit Entwicklerteam:
1 Std * 4 Personen * 130€/Std = 520€
Wöchentliche Absprachen bezüglich der Oberfläche:
1 Std * 6 Personen * 130€/Std = 780€
Allgemeine regelmäßige Anforderungsabsprachen mit Auftraggeber:
5 Std * 1 Person * 130€/Std = 650€

= 8850€ an Kosten für die Fußzeilenkomponente.

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.

$$\frac{100.000\ €}{40\ €/\ Vertrag/\ Jahr} = 2500\ Verträge$$

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

$$\frac{2500\ Verträge}{3\ Jahre} \approx 833\ Verträge * 3\ Jahre * 40\ € \approx 100.000\ €$$

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. Vorteile dieses Deployments ist außerdem, dass es mehrere Versionen und unterschiedliche Stages gibt die zeitgleich laufen. Für jeden Feature-, Develop- und Masterbranch im Git wird jeweils eine eigene Instanz im Kubernetes Cluster erstellt, so dass man jeden Stand auf dem Testsystem testen kann. Dadurch ist die Anwendung besser testbar und neue Versionen können nahtlos veröffentlicht werden.

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 benutzt, 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 halten wird jede Änderung im Frontend in den Localstorage des Browsers geschrieben und zeitgleich per HTTP-JSON-PATCH Aufruf ans 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. Außerdem 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 und das finale Design der Fußzeilenkomponente sind jeweils im Anhang zu finden (s. A.2 Mockup-Design der Fußzeile, A.3 Tatsächliches Design der Fußzeile)

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 Daten, diese können z.B. zu gespeicherten Angeboten oder Tarifkonfigurationen sein, im Cache gespeichert. Dieser ist nur ein temporärer Speicher der die Daten für bis zu 6 Stunden im Arbeitsspeicher hält. Die zweite Stufe ist der Distributed Cache, wird aber im Verlauf dieser Dokumentation Storage genannt. 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 durch den Aufruf der Shutdown-Action im Backend. Dabei werden die Daten einen Tag lang persistiert. 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, in den Storage.
Für die Zustandsverwaltung der Daten im Frontend wird der Pinia-Store benutzt. Dieser ist typsicher, was das Arbeiten mit Typescript erleichtert. Allgemein können mit so einem Store die Daten komponentenübergreifend einheitlich gespeichert werden. In dem OTR werden dort 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 gegeben sind und eventuelle Sitzungsdaten, wenn eine bestehende Sitzung geladen wurde. Außerdem werden berechnete Beiträge sowie eingegebene Daten gespeichert. Auf die Daten aus einer anderen Komponente zugreifen kann man mit sogenannten Gettermethoden, welche im Store definiert werden.

4.5. Entwurf der Geschäftslogik

Die Logik der Benutzeroberfläche und die technische Umsetzung wurde innerhalb des Entwicklerteams durch Prototyping erarbeitet. Dadurch musste für die Fußzeilenkomponente speziell kein eigener Entwurf entstehen, es wurde anhand des Entwurfs für das Gesamtprojekt bearbeitet.
Wie schon in 1.2 Projektziel beschrieben, wird die Oberfläche unter anderem mit Vue.js realisiert. Vue.js großer Vorteil ist es, dass die Oberfläche reaktiv gemacht wird. Das bedeutet, dass wenn sich Daten die mit einem Oberflächenelement verbunden sind ändern, wird dieses Element automatisch aktualisiert. Dies geschieht über das Observer-Designpattern.
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 losgetreten, so dass dann 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. Zu dem 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 Entwicklerteam in den develop-Branch überführt. Bei jedem Push werden per Gitlab Pipeline Unittests im Front- und Backend ausgeführt. Im Backend habe ich die bestehenden Unittests erweitert um neu implementierte Logik zu testen. Zu dem wurde im Frontend ein Linter konfiguriert um einheitliche Codeguidelines und Codestyling durchzusetzen.
Wenn eine Änderung auf den Developbranch gepusht wurde wird diese von einem anderen Entwickler getestet. Außerdem gibt es weitere Mitarbeiter außerhalb des Entwicklerteams die regelmäßig auf dem Testsystem Blackbox-Akzeptanztests durchführen.

5. Implementierungsphase

5.1. Implementierung der Datenstruktur

Die Datenstruktur wie in 4.4 Entwurf der Datenstruktur beschrieben, war schon gegeben. Dementsprechend konnte der 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 habe ich 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 Angebot- bzw. Kopiespeichern-Schaltfläche 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 bzw. Vertriebspartners benötigt. Dementsprechend werden diese Daten mit ans 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 passiert beim Kopiespeichern das Gleiche wie bei der normalen Speicherung, außer dass ein anderer Endpunkt aufgerufen wird. Die Logik im Backend unterscheidet sich dabei allerdings von der Angebotsspeicherung.

Die Angebot- und Antragdrucken 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 Sessionguid, 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 außerdem 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 bereits, 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. Außerdem wird die Logik für die verschiedenen Versicherungstarife aufgeteilt, da die Produkte oder Versicherungstarife verschieden sind. Die unterschiedliche Logik der Tarife wird dynamisch je nach aktivem Tarif benutzt. Das wird durch Dependency Injection ermöglicht. Für die Fußzeilenkomponente ist nur der Managementservice 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, wird sich 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 Bedindungen im Frontend geprüft werden, nochmal Backendseitig vollzogen. Da Angebote und Anträge rechtlich binden 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 reingegeben.
Die PDF-Dateien des Angebots und Antrag kann man sich auch per E-Mail zu senden lassen und aus diesem Grund benutzt die Dokumente-Versenden-Logik den gleichen Programmcode wie das Drucken. Allerdings können auch andere Dokumente versendet werden. Diese müssen beim Aufruf der Routine mit angegeben werden. Die Dokumente werden dann runter geladen und anschließend wird ein Aufruf an den Mailservice

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

6. Projektabschluss

6.1. Abnahme

Durch die regelmäßigen Absprachen und Codereviews durch das Entwicklerteam 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 aufgekommen 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
Ist-Analyse:	**5**	**6**	+1
Analyse vorhandener Codebasen und Funktionalitäten	2	3	+1
Gespräche mit dem Auftragsgeber	3	3	0
Sollkonzept:	**5**	**4**	-1
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
Realisierung:	**31**	**30**	-1
Frontend	**10**	**12**	+2
Entwicklung Oberfläche	4	1	-3
Backend	17	17	0
Dokumentation	19	19	0
Testen & Abnahme	8	8	0
Gesamt	69	69	0

Von dem geplanten Umfang des Teilprojekts wurde alles erfüllt. Zu dem wurde die in 2.2 Abweichung vom Projektantrag erwähnte Angebotskopiespeicherung implementiert.

6.3. Gewonnene Erkenntnisse

Die Umsetzung dieses Projekt 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 zukünftige Erweiterungen problemloser gestalten.
Dazu kommt, dass das erste Mal mit gegebenen Oberflächenmockups gearbeitet wurde, wodurch das Designen und Entwerfen der Oberfläche erleichtert wurde. Außerdem ist durch die Konzeption des Gesamtprojekt 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. Außerdem 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. Außerdem sind kleinere Refactorings im Backend sowie ein Refactoring der Datenzustandsverwaltung im Frontend geplant.

Literaturverzeichnis

A. Anhänge

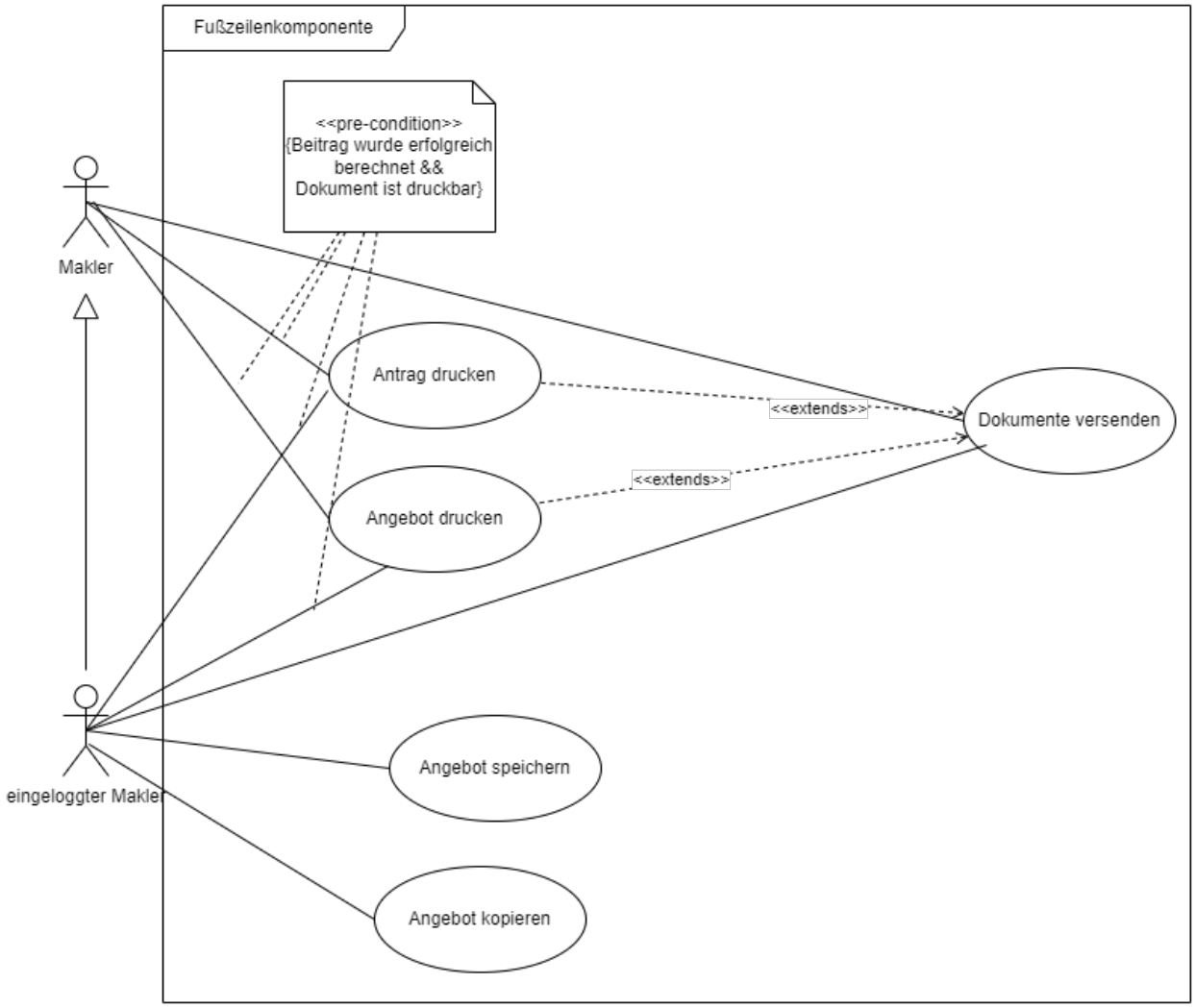

Abbildung 1: Anwendungsfalldiagramm der Fußzeilenkomponente

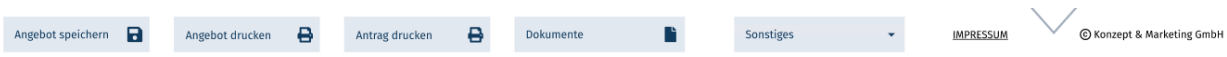

Abbildung 2: Mockup-Design der Fußzeile

Abbildung 3: Tatsächliches Design der Fußzeile