



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:

Jakob Schumann
Goethestraße 40
30169 Hannover

Ausbildungsbetrieb:

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

Inhaltsverzeichnis

1. Einleitung	1
1.1. Projektumfeld	1
1.2. Projektziel	1
1.3. Projektbegründung	3
1.4. Projektschnittstellen	3
1.5. Projektabgrenzung	4
2. Projektplanung	5
2.1. Projektphasen	5
2.2. Abweichung vom Projektantrag	5
2.3. Ressourcenplanung	6
2.4. Entwicklungsprozess	6
3. Analysephase	7
3.1. Ist-Analyse	7
3.2. Wirtschaftlichkeitsanalyse	7
3.3. "Make or Buy Entscheidung	7
3.4. Projektkosten	7
3.5. Amortisationsdauer	7
3.6. Anwendungsfälle	7
3.7. Anforderungen	7
4. Entwurfsphase	8
4.1. Zielplattform	8
4.2. Architekturdesign	8
4.3. Entwurf der Benutzeroberfläche	8
4.4. Datenmodell	9
4.5. Geschäftslogik	9
4.6. Maßnahmen zur Qualitätssicherung	9
5. Implementierungsphase	10
5.1. Implementierung der Datenstruktur	10
5.2. Implementierung der Benutzeroberfläche	10
5.3. Implementierung der Geschäftslogik	11
6. Abnahmephase	14
7. Einführungsphase	15

8. Dokumentation	16
9. Fazit	17
9.1. Soll-Ist-Vergleich	17
9.2. Gewonnene Erkenntnisse	17
9.3. Ausblick	17
Literaturverzeichnis	I
Abbildungsverzeichnis	II
A. Anhänge	III

Abkürzungsverzeichnis

K&M Konzept und Marketing GmbH

SPA Single-Page-Application

OTR Onlinetarifrechner

GUID Globally Unique Identifier

TG Tarifrechner-Gateway

TRG Tarifrechner-REST-Gateway

DTO Data Transfer Object

CI/CD Continuous Integration/Continuous Delivery

Glossar

fetch Asynchroner Rest-Aufruf an eine Schnittstelle. 10

HTML Hypertext markup language. 10

HTTP Hypertext Transfer Protocol; Protokoll für Datenübertragung. 10

npm Ein Paketmanager für Node.js Pakete. 9

Pinia-Store Eine Datenzustandsverwaltung für Vue.js. 9, 10

TypeScript Eine typsichere Erweiterung von JavaScript. 9

Vue.js 3 Ein clientseitiges Javascript-Framework nach dem Model-View-ViewModel-Entwurfsmuster für das Erstellen von Webanwendungen. 9, 10

Vuetify Eine Vue.js Oberflächenkomponentenbibliothek. 9

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 und diese auch Komplett von der Annahme bis hin zur Schadensregulierung verwaltet.

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 Online Tarifrechnern, 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 Onlinetarifrechner (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 kann. 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 viele Schnittstellen zu umliegenden, firmeninternen Systemen, sowie eine Cachespeicherung besitzt. Das Frontend besteht aus Vue.js 3.0 mit Designvorlagen von Vuetify, sowie einer eigenen Oberflächenkomponentenbibliothek welche Vuetifyelemente kapselt und im Corporatedesign zur Verfügung stellt. Im Frontend benutzen wir Typescript als Programmiersprache.

Die Kommunikation zwischen Front- und Backend wird durch REST-API Aufrufe realisiert. Um die Daten synchron zu halten wird jede Änderung im Frontend in einen Speicher, welcher den Localstorage des Browsers benutzt, geschrieben und zeitgleich per HTTP-JSON-PATCH API-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. Alle Schnittstellen sowie Data Transfer Object (DTO) werden beim Bauen des Backends automatisch für das Frontend generiert. 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, also wird das Projekt vorher als Dockerimage gebaut um es dann in einem Container laufen zu lassen. Für jeden Feature-, Develop- und Master-branch im Git wird jeweils eine eigene Instanz im Kubernetes Cluster erstellt, so dass man jeden Stand auf dem Testsystem testen kann.

Bei einem OTR hat ein Makler die Möglichkeit Angaben zu einem Versicherungstarif zu treffen und sich anhand dessen Beiträge ausrechnen zulassen. Außerdem sind hilfreiche 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 Angebotes sollen alle im OTR eingegebenen Daten im sogenannten Maklerportal für den angemeldeten Makler gespeichert werden. Das Maklerportal ist eine Webseite an der sich Makler anmelden können wo sie den OTR aufrufen können, Übersichten über ihr Kundenbestand und noch vieles mehr. Zu dem können dort auch gespeicherte Angebote eingesehen, bearbeitet und mit einem OTR aufgerufen 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 Knöpfe abgebildet, einen um ein Angebot zu drucken, der 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. Ein Dokument soll dann gedruckt werden können, wenn Beiträge erfolgreich berechnet wurden. Dafür müssen alle berechnungsrelevanten Angaben getroffen worden sein. Außerdem kann es sein, dass

für einen Tarif der Angebot- oder Antragsdruck deaktiviert ist. Die Knöpfe sind bis das geschehen ist deaktiviert und werden reaktiv aktiviert. Dementsprechend sind die Schaltflächen aktiviert, oder nicht aktiviert. Das Drucken ist dabei eigentlich das Herunterladen eines PDF-Dokumentes, damit der Makler dieses auch per E-Mail verschicken oder selber ausdrucken kann. 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. Dazu muss noch eine oder mehrere E-Mails angegeben werden und dann werden die Dokumente per E-Mail in einem ZIP-Archiv versendet. Die Dokumente beinhalten rechtliche Informationen und der 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 Frontend nur eingebunden werden musste.

1.3. Projektbegründung

Die Motivation hinter diesem Projekt begründet sich in der einfachen Erweiterbarkeit für neue Versicherungsprodukte sowie eine Überholung des Aussehens des OTR. Allgemein 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 die Fußzeile bietet sich dafür gut an, da sie unabhängig von dem Kontext der Rest der 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 Online Tarifrechtern 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. Das TRG speichert Daten in eine MS SQLServer Datenbank, während der neue OTR keine direkte Anbindung an eine Datenbank hat.

Eine weitere Schnittstelle die von der Fußzeilenkomponente genutzt werden 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. Der OTR wird vor allem von Maklern benutzt um Tarife für ihre Endkunden auszurechnen.

1.5. Projektabgrenzung

Das bearbeitete Projekt ist hierbei nur die Fußzeile mit den Schaltflächen einschließlich ihrer Funktionalitäten. Eine Benutzeroberflächenkomponentenbibliothek ist gegeben, aus welcher die Schaltflächen eingebunden werden. Genauso ist die vorher beschriebene Synchronisation 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 einschließlich den 18.03.2022 durchgeführt.

Achtung,
richtiges
Datum

2.1. Projektphasen

Projektphasen	Zeit in Stunden
tägliche Absprachen mit Entwicklungsteam + Auftragsgeber	1
wöchentliche Absprache zur Oberfläche	2
Ist-Analyse:	5
Analyse vorhandener Codebasen und Funktionalitäten	3
Gespräche Auftragsgeber	4
Sollkonzept:	5
Planung des Soll-Zustands Projekt allgemein	2
Planung des Soll-Zustands Fußzeilenfunktionalität	1
Abstimmung + Sichtung Workflows und Oberflächenentwürfe	1
Realisierung:	31
Frontend	10
Knöpfe eingebunden + Styling	2
Funktionalität Backend Anbindung, optionales Rendering	6
Eventhandling, Label mit i18n	2
Backend	17
Angebotspeicherung	8
Kopiespeicherung	2
Dokumentengenerierung	7
Entwicklung Oberfläche	1
Dokumentation	19
Testen + Abnahme	8
Gesamt	69

2.2. Abweichung vom Projektantrag

Alle im Projektantrag beschriebenen Funktionalitäten wurden umgesetzt. Hinzu kam die Kopie-
speicherung eines Angebotes. Außerdem war schon mehr Grundgerüst im Form von bestehenden
Serviceprojekten und Schnittstellen gegeben, wodurch nur noch die reine Logik implementiert
werden musste. Die UI-Elemente mussten neu erstellt werden oder ggf. erweitert oder angepasst
werden.

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, 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 Programmteile zum Testen zur Verfügung gestellt.

Dieses Vorgehensmodell wurde gewählt damit man die Entwicklung schnell an neue Anforderung der Kunden, bzw. der Stakeholder anpassen kann.

besseres
Wort
finden

3. Analysephase

3.1. Ist-Analyse

Die K&M bietet OTR für Makler an. Zur Zeit gibt es mehrere Generationen an Rechnern. Die älteste ist mit PHP umgesetzt und hat eine altmodische Benutzeroberfläche. Eine neuere Tarifrheberbasis ist ähnlich wie diese Neuentwicklung aufgebaut auf einem .NET-Backend und einem Vue.js Frontend. Allerdings stellte sich bei dem Einführen neuer Tariflinien heraus, dass diese sich nicht einfach in diesen OTR einbinden ließ.

Demnach existieren funktionale OTR, die beim Vertrieb von Versicherungsprodukten helfen, allerdings sind diese entweder für den Endnutzer nicht anschaulich oder modern und/oder für den Entwickler schwierig zu warten oder zu erweitern.

3.2. Wirtschaftlichkeitsanalyse

3.3. "Make or Buy Entscheidung

3.4. Projektkosten

3.5. Amortisationsdauer

3.6. Anwendungsfälle

3.7. Anforderungen

4. Entwurfsphase

Der OTR ist schon lange in Entwurf und wurde

4.1. Zielplattform

Das gesamte Projekt soll in einem Container innerhalb eines Kubernetes Cluster laufen. Demnach ist die Zielplattform eine Linuxdistribution. Durch Kubernetes besteht ein Loadbalancing, falls zu viele Anfragen gleichzeitig reinkommen. Außerdem wird durch die Gitlab Pipeline **cicd!** (**cicd!**) ermöglicht und in Kombination mit Helm wird das Veröffentlichen neuer Versionen stark vereinfacht.

4.2. Architekturdesign

Die Backend Architektur besteht aus Microservices nach dem Domain-Driven Design-Pattern. Außerdem wird für die Datenpersistenz das Repository-Pattern benützt. Im Frontend werden Komponenten benutzt, welche ihre Daten über eine zentrale Datenzustandsverwaltung beziehen. Die Kommunikation zwischen Front- und Backend geschieht über REST-Aufrufe. Daten aus dem Frontend werden per JSONPatch Protokoll und demnach über die HTTP-Patch methode übermittelt. Dadurch können Änderungen auch kleinteilig synchronisiert und persistiert werden.

4.3. Entwurf der Benutzeroberfläche

Der designseitige Entwurf der Benutzeroberfläche wurde durch die Marketing-Abteilung per Mockups übernommen. Diese mussten allerdings nicht pixelperfect umgesetzt werden, sondern konnten in Kombination mit dem Corporate-Design-Vorlagen als Richtlinie genutzt werden. Außerdem gab es wöchentliche Absprache sowie Absprachen mit einem externen Dienstleister, welcher Experte in UI/UX ist, in denen speziell das Design abgestimmt wurde. Die Logik der Benutzeroberfläche und die technische Umsetzung wurde innerhalb des Entwicklerteams durch Prototyping erarbeitet. Dadurch musste für die Fußzeilenkomponente speziell keine eigene Logik oder Design entworfen werden, es wurde anhand des Entwurfs für das Gesamtprojekt bearbeitet.

4.4. Datenmodell

4.5. Geschäftslogik

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 merged. Automatisch laufen durch die Gitlab Pipeline Unittests im Front- und Backend. Im Backend habe ich die bestehenden Unittests erweitert um neu implementierte Logik zu testen. Zudem wurde im Frontend ein Linter konfiguriert.

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

5.2. Implementierung der Benutzeroberfläche

Die Benutzeroberfläche wird mit Vue.js 3, Vuetify und TypeScript realisiert. Außerdem gibt es eine interne Komponentenbibliothek als npm-Paket, welche Vuetify-Elemente kapselt und im Corporate Design und mit geringfügiger aber erweiterbarer Funktionalität anbietet. Vue.js 3 großer Vorteil ist es, dass es die Oberfläche reaktiv macht. Das bedeutet, dass wenn sich Daten die mit einem Oberflächenelement verbunden sind ändern, wird dieses Element automatisch aktualisiert. Dies geschieht über das Observer-Designpattern. Vuetify, bzw. die Komponentenbibliothek hilft bei einem einheitlichem Design und erleichtert die Einbindung. Die durch TypeScript bereit gestellte Typsicherheit macht das Arbeiten im Team, das Benutzen von Bibliotheken, vorausgesetzt diese haben Typescriptdatentypen deklariert sowie die Wartung und zukünftige Erweiterungen einfacher.

Für die Zustandsverwaltung der Daten im Frontend wird der Pinia-Store benutzt, eine Erweiterung des Vue.js 3-Stores. Dieser ist typsicher, was das Arbeiten mit Typescript erleichtert. Allgemein können mit so einem Store die Daten Komponenten übergreifend einheitlich gespeichert werden. In dem OTR werden dort wichtige Daten welche das Backend bei der Initialisierung liefert gespeichert. Dazu gehören unter anderem Konfigurationen wie der OTR aufgebaut werden soll welche Eingabemöglichkeiten gegeben sind und eventuelle Sitzungsdaten, wenn eine bestehende Sitzung geladen wurde. Außerdem werden berechnete Beiträge sowie eingegebene Daten gespeichert. Alle eingegebenen Daten werden automatisch beim Speichern im Store gleichzeitig ans Backend geschickt. Es gibt mehrere Teilstores, einen allgemeingültigen Store, in welchen Daten gespeichert werden welche tarifunabhängig sind und ein Store für den jeweilig aktiven Tarif. Auf die Daten aus einer anderen Komponente zugreifen kann man mit sogenannten Gettermethoden, welche im Store definiert werden.

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 Erfolgsmeldung bei einer Aktion, ausgegeben wird.

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 der Button 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 Angebot- bzw. Kopiespeichern-Schaltfläche soll nur vorhanden sein, wenn der OTR über das Maklerportal aufgerufen wurde. Für so etwas bietet Vue.js 3 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 den Knopf wird ein fetch Aufruf ans Backend an den Management-Controller an die SaveOrder bzw SaveOrderCopy Action gemacht. Dieser ist durch automatisch generierte Klassen anhand der Schnittstellendefinition des Backends gekapselt. Diese kümmern sich um Fehlerbehandlung anhand des HTTP-Statuscodes, erwarten die Argumente als richtigen Datentyp und geben die Response als eigenen Datentyp zurück. Ein Angebot wird anhand einer Globally Unique Identifier (GUID) gespeichert, welche die aktuelle Sitzung des OTR repräsentiert. Außerdem wird die Identifikationsnummer des angemeldeten Maklers, bzw. Vertriebspartners benötigt. Dementsprechend werden diese Daten mit ans Backend übergeben. Dieses kann sich anhand der Session die Angebotsdaten aus dem Cache laden und diese dann an das TRG zum Speichern weiterschicken. Auf der Seite des Frontends passiert beim Kopiespeichern das Gleiche wie bei der normalen Speicherung, außer dass ein anderer Endpunkt aufgerufen wird.

Zu
Spezifisc
mit den
Namen?

Die Angebot- und Antragdruck-Schaltflächen sind zwar immer vorhanden, sind allerdings nicht immer aktiviert, bzw. benutzbar. Diese Logik setzt sich aus zwei Bedingungen zusammen. Zum einen muss überhaupt für den aktuellen Tarif der Antrags- bzw. Angebotsdruck aktiviert sein. Zum anderen ist das Drucken erst möglich, wenn ein Beitrag berechnet werden konnte. Diese Bedingungen werden im Pinia-Store überprüft und in den Schaltflächenkomponenten in dem disabled-Attribut benutzt. Beim Drücken der Buttons wird ein Backend Aufruf an die Drucklogik gestartet. Dieser erwartet die Sessionguid, damit sich das Backend die benötigten Daten aus dem Cache laden kann. Zurückgeliefert wird ein PDF-Dokument als Filestream . Das wird dann im Frontend aufbereitet, so dass automatisch ein PDF-Dokument heruntergeladen wird. Da die Logik hierbei im Frontend zwischen dem Antrag und Angebot sich nur im aufgerufenem Endpunkt unterscheidet, ist die Logik zusammengefasst und der Aufruf des Backends kann als Action per Parameter reingegeben werden.

Prüfen
wie
genau?

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

Die Geschäftslogik, bzw. das Backend ist eine .NET 6.0 API. Als Programmiersprache wird dementsprechend C# 10 benutzt. Das Backend persistiert Daten in einem Storage oder Cache. Da die Kommunikation zwischen dem Backend und den anliegenden Systemen größtenteils über XML

writer
lock,cach
repository
pattern

läuft, werden die gegebenen XML-Schemata eingelesen und beim Bauen des Backends als C# Klassen generiert. Dadurch können die Daten einheitlich und einfach geparkt und ausgelesen werden. Allerdings kann es trotzdem vorkommen, dass Daten in anderen Datentypen verarbeitet werden. Um diese dann in das besagte XML-Format zu stecken wird der Automapper genutzt, welchen man nur konfigurieren muss um die Daten in der jeweils andere Klasse speichern zu können.

besser
formulieren

Das Backend ist im Domain-Driven Design aufgebaut. Dementsprechend gibt es mehrere Services, welche jeweils in mehrere Projekte aufgeteilt sind. Unter anderem gibt es im Backend Schnittstellen für Anträge, Beitragsberechnung, Tarifkonfigurationen und allgemeine Logik unter Management. Jede dieser Schnittstellen haben 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 sie sich dort unterscheiden kann. Die unterschiedliche Logik der Tarife wird nach Strategy Pattern dynamisch je nach aktivem Tarif benutzt und per Dependency Injection Pattern reingegeben. Für die Fußzeilenkomponente ist nur der Managementservice interessant, da dort allgemeingültige und tarifunspezifische Logik zu finden ist.

nicht
wirklich
strategy?
prüfen?

Zum Speichern und Kopieren von Angeboten wird die Session als **guid!** (**guid!**) benötigt. Anhand dieser Sessionguid, wird sich die Konfiguration und die gespeicherten Angebotsdaten aus dem Repositorycache 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 Repositorystorage 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 Angebotes, 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 testweise eine Beitragsberechnung durchgeführt. Erst wenn dies 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.

Da man sich genau diese PDF-Dateien auch per E-Mail zu senden kann, benutzt die Dokumenten-Versenden-Logik den gleichen Programmcode wie das Drucken. Allerdings können auch andere

Dokumente versendet werden. Diese müssen beim Aufruf der Action mit angegeben werden. Die Dokumente werden dann runter geladen und anschließend wird ein Aufruf an den Mailservice getätigt, welcher diese Dokumente per angegebene E-Mail versendet.

6. Abnahmephase

7. Einführungsphase

8. Dokumentation

9. Fazit

9.1. Soll-Ist-Vergleich

9.2. Gewonnene Erkenntnisse

9.3. Ausblick

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

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

Abbildungsverzeichnis

A. Anhänge