



Abschlussprüfung Winter 2022

Fachinformatiker/Fachinformatikerin (VO 2020) Fachrichtung:  
Anwendungsentwicklung, Leipzig FIAE 4 (AP T2V1)

Dokumentation zur betrieblichen Projektarbeit

## **ReklaTool**

**Entwicklung einer Webanwendung zur Abfrage und Anzeige  
von Kalkulationsvorgängen aus dem Kfz-Bereich  
nach dem Client-Server-Prinzip**

Abgabetermin: Leipzig, den 23.11.2022

**Prüfungsbewerber:**

Ingolf Schieck  
Identnummer: 661537  
Hähnelstraße 21  
04177 Leipzig



**Ausbildungsbetrieb:**

IcamSystems GmbH  
Käthe-Kollwitz-Straße 84  
04109 Leipzig

---

## Urheberrechtserklärung

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

## Gender-Erklärung

Aus Gründen der besseren Lesbarkeit hat sich der Autor dieser Arbeit für die Verwendung des generischen Maskulinums entschieden. Sämtliche Formulierungen, z.B. Mitarbeiter, Nutzer, etc., gelten gleichermaßen für alle Geschlechter.

# Winterprüfung 2022

## **Ausbildungsberuf**

Fachinformatiker/Fachinformatikerin (VO 2020) Fachrichtung:  
Anwendungsentwicklung

## **Prüfungsbezirk**

Leipzig FIAE 4 (AP T2V1)

Ingolf Schieck

Identnummer: 661537

E-Mail: ingolf.schieck@outlook.com, Telefon: +49 176 78647726

Ausbildungsbetrieb: IcamSystems GmbH

Projektbetreuer: Danny Sotzny

E-Mail: danny.sotzny@icamsystems.de, Telefon: +49 341 98999 308

## **Thema der Projektarbeit**

ReklaTool – Entwicklung einer Webanwendung zur Abfrage und Anzeige von  
Kalkulationsvorgängen aus dem Kfz-Bereich nach dem Client-Server-Prinzip

# 1 Thema der Projektarbeit

ReklaTool – Entwicklung einer Webanwendung zur Abfrage und Anzeige von Kalkulationsvorgängen aus dem Kfz-Bereich nach dem Client-Server-Prinzip

## 2 Geplanter Bearbeitungszeitraum

Beginn: 10.10.2022

Ende: 28.10.2022

## 3 Ausgangssituation

Projektbeschreibung und Umfeld

Die IcamSystems GmbH bietet Softwarelösungen im Bereich der Kfz-Schadensregulierung an. Sie ist Teil eines Firmenverbundes mit ca. 160 Mitarbeitern. Zu den Kunden zählen unter anderem Versicherungsunternehmen, Sachverständigenorganisationen, Prüfdienstleister und Schadensteuerungsgesellschaften.

Zum Leistungsspektrum gehört die automatisierte Prüfung von Gutachten und Kostenvoranschlägen, welche durch Gutachter oder Partnerwerkstätten erstellt wurden.

Im Fall, dass ein Kunde nicht mit dem Ergebnis dieser Prüfung einverstanden ist, hat er die Möglichkeit eine Reklamation an die IcamSystems GmbH zu senden. Um den Sachbearbeitern die Möglichkeit zu geben, sich schnell eine Übersicht über den beanstandeten Vorgang zu verschaffen, soll eine Webanwendung für das Firmenintranet erstellt werden.

Diese fragt Daten mittels eines Webservices von verschiedenen Datenbanken ab und stellt diese aufbereitet in übersichtlicher Form zur Verfügung.

Ist-Analyse

Die Vorgangsprüfung erfolgt über das automatisierte Prüfregelwerk ClaimsGuard (kurz CG).

Darin prüft ein individuell erstelltes Regelwerk die, z.B. von Werkstätten, erstellten Kostenvoranschläge und Gutachten. Für die individuellen Parameter jedes Fahrzeugtyps greift der CG auf die recherchierten Fahrzeugdaten der Vehicle Information Database (kurz VID) zurück. Die VID bietet für jedes Bauteil Informationen zur Beschaffenheit und Verarbeitung. Bei der Plausibilitätsprüfung durch den CG entsteht ein detaillierter Prüfbericht, in dem die angewendeten Prüfregeln gelistet sind und ggf. Diskrepanzen aufgeführt werden.

Sollte der Kunde das Prüfergebnis beanstanden, so hat er die Möglichkeit eine Reklamation an die IcamSystems GmbH zu senden.

Aktuell können nur die Projektverantwortlichen selbst, mittels des jeweiligen Aktenzeichens der Reklamation, eine händische Suche in mehreren Datenbanken und dem Regelwerk des CG durchführen, um den Vorgang nachzuvollziehen. Es soll für die Bewertung des Vorgangs ersichtlich sein, welche Regeln ausgelöst wurden.

Diese Tätigkeit ist sehr zeitaufwändig und soll für eine schnellere Abfrage vereinfacht und standardisiert werden.

## 4 Projektziel

Ziel ist eine Webanwendung mit einer funktionalen Benutzeroberfläche, welche Daten von einem Backend-Service abfragt, um diese strukturiert und übersichtlich darzustellen. Reklamationen sollen so schneller bearbeitet werden können und die händische Suche nach allen Teilinformationen überflüssig machen.

### Projektanforderungen

- Eingabe von vorgangsspezifischen Aktenzeichen als Suchwörter
- Möglichkeit einer Schnellsuche ohne Einzelprüfbericht und Regelauslösungen
- Abfrage von externen Datenbanken und Aufbereitung der Daten zum Vorgang
- Auswahl eines Vorgangs zum gesuchten Aktenzeichen
- Aufbereitung und Anzeige des ausgewählten Vorgangs in folgende Bereiche:
  - o Allgemeine Übersicht zur Kalkulation
  - o Übersicht der kalkulierten Positionen
  - o Informationen aus der Fahrzeugdatenbank VID
  - o Ausgelöste Regeln für den Vorgang
- Download des Einzelprüfberichts zum Vorgang
- Download der Kalkulation als strukturierte Datei
- Authentifizierung per Identity Server mittels OAuth2
- Suche darf nicht länger als 7 Sekunden ohne Reaktion bleiben
  - o Abfrage aller Daten dauert in der Regel über 30 Sekunden
  - o Lösungsansatz: Bereitstellen von Teilergebnissen

### Umsetzung

Das Projekt wird anhand der Phasen des Wasserfallmodells umgesetzt. In einzelnen Phasen, besonders bei Analyse und Entwurf, ist trotzdem die Rücksprache mit dem Fachbereich vorgesehen.

Um die spätere Pflege und Weiterentwicklung durch das Team zu gewährleisten, wird das Projekt in JavaScript und C# auf Basis der SOLID-Prinzipien umgesetzt. Zusammen mit dem .NET-Framework ist dies der Standard für den Großteil der firmeninternen Projekte. Als Zieltechnologie wurde das etablierte Webframework ASP.Net Core MVC ausgewählt. Zur weiteren Sicherung der Wartbarkeit und Erweiterbarkeit wird das Projekt im MVC-Pattern realisiert. Dieses wird u.a. durch Services im Bereich von http-Anfragen und Caching erweitert.

Die Weboberfläche soll mit dem KendoUI-Framework von Telerik umgesetzt werden, da dieses eine Vielzahl von Komponenten zur Darstellung von Daten mitbringt und bereits Lizenzen in der

Firma zur Verfügung stehen.

Bereits während der Implementierung soll die Produktqualität durch Unit- und Integrationstests sichergestellt werden.

## 5 Zeitplanung

### Grobplanung

Analyse 8h  
Entwurf 16h  
Implementierung und Tests 40h  
Abnahme 3h  
Dokumentation 13h

Gesamt 80h

### Detailplanung

#### Analyse (8h)

Ist-Analyse 1h  
Make-Or-Buy-Analyse 1h  
Ermittlung von Use-Cases zusammen mit dem Fachbereich und Erstellung eines  
Use-Case-Diagramms 1h  
Soll-Konzept 2h  
Unterstützung der Fachabteilung bei der Erstellung des Lastenhefts 1h  
Wirtschaftlichkeitsbetrachtung mit Amortisationsrechnung 1h  
Zeit- und Ablaufplanung 1h

#### Entwurf (16h)

Planung der Projektstruktur (Versionskontrolle, Ordnerstruktur, etc.) 3h  
Erstellung von Mock-Ups der Weboberfläche 4h  
Entwurf der Systemabgrenzung, Erstellung eines Systemkontextdiagramms 2h  
Entwurf des Programmflusses, Erstellung eines Sequenzdiagramms 4h  
Entwurf der Klassen, Erstellung eines Klassendiagramms 3h

#### Implementierung und Tests (40h)

Erstellung und Konfigurierung des C#-Projekts 3h  
Erstellen der Modell-Klassen 3h  
Erstellen des http-Service, inkl. Tests 8h  
Erstellen des Cache-Service, inkl. Tests 6h  
Erstellung und Ausführen von Integrationstests 6h  
Implementierung der User-Authentifizierung 6h  
Erstellung der Weboberfläche, inkl. Datenbindung und Interaktionen 8h

#### Abnahme (3h)

Codereview und technische Abnahme 2h  
Abnahme durch den Fachbereich 1h

## Dokumentation (13h)

Erstellen der Nutzerdokumentation 4h

Erstellen der Entwicklerdokumentation 1h

Projektbewertung 1h

Erstellen der Projektdokumentation 7h

## 6 Anlagen

keine

## 7 Präsentationsmittel

Laptop

Beamer

Presenter

## 8 Hinweis!

Ich bestätige, dass der Projektantrag dem Ausbildungsbetrieb vorgelegt und vom Ausbildenden genehmigt wurde. Der Projektantrag enthält keine Betriebsgeheimnisse. Soweit diese für die Antragstellung notwendig sind, wurden nach Rücksprache mit dem Ausbildenden die entsprechenden Stellen unkenntlich gemacht.

Mit dem Absenden des Projektantrages bestätige ich weiterhin, dass der Antrag eigenständig von mir angefertigt wurde. Ferner sichere ich zu, dass im Projektantrag personenbezogene Daten (d. h. Daten über die eine Person identifizierbar oder bestimmbar ist) nur verwendet werden, wenn die betroffene Person hierin eingewilligt hat.

Bei meiner ersten Anmeldung im Online-Portal wurde ich darauf hingewiesen, dass meine Arbeit bei Täuschungshandlungen bzw. Ordnungsverstößen mit „null“ Punkten bewertet werden kann. Ich bin weiter darüber aufgeklärt worden, dass dies auch dann gilt, wenn festgestellt wird, dass meine Arbeit im Ganzen oder zu Teilen mit der eines anderen Prüfungsteilnehmers übereinstimmt. Es ist mir bewusst, dass Kontrollen durchgeführt werden.

## IT-Berufe – Nachweis über die Durchführung der betrieblichen Projektarbeit

### Abschlussprüfung IT-Berufe

<b>Name, Vorname:</b>		<b>Berufsbezeichnung/Fachrichtung:</b>
Schieck, Ingolf		Fachinformatiker Anwendungsentwicklung
<b>Prüflings-Nr.:</b>		<b>Maximaler Zeitaufwand:</b>
661537		80 Stunden

Datum	Anzahl der Stunden	ausgeführte Tätigkeiten
10.10.2022	3	Ist-Analyse, Make-Or-Buy-Analyse, Erstellung Use-Cases
10.10.2022	2	Erstellung Soll-Konzept, Anforderungsanalyse
11.10.2022	3	Lastenheft, Wirtschaftlichkeitsanalyse, Zeitplanung
11.10.2022	3	Planung Projektstruktur, Erstellung Pflichtenheft
12.10.2022	4	Erstellung von Mock-Ups
13.10.2022	6	Erstellung Systemkontext- und Sequenzdiagramm
14.10.2022	3	Entwurf und Erstellung Klassendiagramm
17.10.2022 - 21.10.2022	30	Implementierung und Testing
24.10.2022 - 26.10.2022	14	Implementierung, Testing, Abnahme, Projektbewertung

### Anfertigung der Dokumentation:

Datum	Anzahl der Stunden	
27.10.2022	4	Erstellung Nutzerdokumentation
27.10.2022	1	Erstellung Entwicklerdokumentation
28.10.2022	7	Erstellung Projektdokumentation
<b>Gesamt:</b>	80	

Ich versichere, dass ich die Projektarbeit einschließlich Dokumentation ohne fremde Hilfe und nur mit den angegebenen Hilfsmitteln erstellt habe. Das Projekt hat stattgefunden.

<p><i>Leipzig 28.10.22</i></p> <p>Ort, Datum</p>	<p></p> <p>Unterschrift Prüfling</p>	<p><i>[Signature]</i></p> <p>Unterschrift Betreuer</p>
--	--------------------------------------	--



## Geschäftsfeld Aus- und Weiterbildung

### Erklärung des Prüfungsteilnehmers / der Prüfungsteilnehmerin

---

Ich versichere durch meine Unterschrift, dass ich das Projekt und die dazugehörige Dokumentation selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe. Die Arbeit hat in dieser Form keiner anderen Prüfungsinstitution vorgelegen.

Leipzig, 28.10.2022

---

Ort, Datum

Unterschrift des Prüfungsteilnehmers

### Erklärung des Ausbildungsbetriebes / Praktikumsbetriebes

---

Wir versichern, dass das Projekt wie in der Dokumentation dargestellt, in unserem Unternehmen realisiert worden ist.

Leipzig, 28.10.2022

---

Ort, Datum

Stempel und Unterschrift des  
Ausbildungs-/ Praktikumsbetriebes

## Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>III</b>
<b>Tabellenverzeichnis</b>	<b>III</b>
<b>Listings</b>	<b>III</b>
<b>Abkürzungsverzeichnis</b>	<b>IV</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Projektumfeld . . . . .	1
1.2 Projektziel . . . . .	1
1.3 Projektbegründung . . . . .	1
1.4 Projektschnittstellen . . . . .	2
1.5 Projektabgrenzung . . . . .	2
<b>2 Projektplanung</b>	<b>3</b>
2.1 Projektphasen . . . . .	3
2.2 Ressourcenplanung . . . . .	3
2.3 Entwicklungsprozess . . . . .	3
<b>3 Analysephase</b>	<b>4</b>
3.1 Ist-Analyse . . . . .	4
3.2 Wirtschaftlichkeitsanalyse . . . . .	5
3.2.1 „Make or Buy“-Entscheidung . . . . .	5
3.2.2 Projektkosten . . . . .	5
3.2.3 Amortisation . . . . .	6
3.3 Nicht-monetärer Nutzen . . . . .	7
3.4 Anwendungsfälle . . . . .	7
3.5 Qualitätsanforderungen . . . . .	7
3.6 Lastenheft . . . . .	7
<b>4 Entwurfsphase</b>	<b>8</b>
4.1 Zielplattform . . . . .	8
4.2 Architekturdesign . . . . .	8
4.3 Entwurf der Benutzeroberfläche . . . . .	9
4.4 Datenmodell . . . . .	9
4.5 Geschäftslogik . . . . .	10
4.6 Maßnahmen zur Qualitätssicherung . . . . .	10
4.7 Pflichtenheft . . . . .	11
<b>5 Implementierungsphase</b>	<b>12</b>

5.1	Konfiguration . . . . .	12
5.2	Authentifizierung . . . . .	12
5.3	Implementierung der Datenstrukturen . . . . .	13
5.4	Implementierung der Benutzeroberfläche . . . . .	13
5.5	Implementierung der Geschäftslogik . . . . .	13
<b>6</b>	<b>Abnahmephase</b>	<b>14</b>
<b>7</b>	<b>Einführungsphase</b>	<b>14</b>
<b>8</b>	<b>Dokumentation</b>	<b>14</b>
<b>9</b>	<b>Fazit</b>	<b>15</b>
9.1	Soll-/Ist-Vergleich . . . . .	15
9.2	Lessons Learned . . . . .	15
9.3	Ausblick . . . . .	16
<b>A</b>	<b>Anhang</b>	<b>i</b>
A.1	Detaillierte Zeitplanung . . . . .	i
A.2	Ressourcen . . . . .	ii
A.3	Break-Even-Point . . . . .	iii
A.4	Lastenheft (Auszug) . . . . .	iii
A.5	Aktivitätsdiagramm . . . . .	iv
A.6	Use-Case-Diagramm . . . . .	v
A.7	Systemkontextdiagramm . . . . .	vi
A.8	MVC-Entwurfsmuster . . . . .	vi
A.9	Sequenzdiagramm . . . . .	vii
A.10	Pflichtenheft (Auszug) . . . . .	viii
A.11	Klassendiagramm . . . . .	ix
A.12	Datenmodell . . . . .	x
A.13	DTO (Ausschnitt) . . . . .	xi
A.14	Oberflächenentwurf . . . . .	xii
A.15	View (Auszug) . . . . .	xii
A.16	Screenshots der Anwendung . . . . .	xiv
A.17	Übergabeprotokoll . . . . .	xvi
A.18	Entwicklerdokumentation (Auszug) . . . . .	xvii
A.19	Benutzerdokumentation (Ausschnitt) . . . . .	xx

## Abbildungsverzeichnis

1	Break-Even-Point . . . . .	iii
2	Aktivitätsdiagramm . . . . .	iv
3	Use-Case-Diagramm . . . . .	v
4	Systemkontextdiagramm . . . . .	vi
5	MVC-Entwurfsmuster . . . . .	vi
6	Sequenzdiagramm . . . . .	vii
7	Klassendiagramm . . . . .	ix
8	XML-Modell für Antwort . . . . .	x
9	XML-Modell für Anfrage . . . . .	x
10	Mockup der Benutzeroberfläche . . . . .	xii
11	Eingabe des Aktenzeichens und Auswahl des Typs (Ausschnitt) . . . . .	xiv
12	Liste der Vorgänge zum Aktenzeichen . . . . .	xiv
13	Darstellung der Daten in einer Tabelle . . . . .	xv
14	Integrierte PDF-Anzeige . . . . .	xv
15	Ansicht des ReklaTool . . . . .	xx

## Tabellenverzeichnis

1	Zeitplanung . . . . .	3
2	Kostenaufstellung . . . . .	5
3	Zeitersparnis . . . . .	6
4	Soll-/Ist-Vergleich . . . . .	15

## Listings

1	ResponseModel.cs – XML-Annotationen . . . . .	xi
2	View-Komponenten – Checkbox und Auswahlliste . . . . .	xii

## Abkürzungsverzeichnis

<b>Ajax</b>	Asynchronous JavaScript and XML
<b>API</b>	Application Programming Interface
<b>BPMS</b>	Business Process Management Software
<b>bzw.</b>	beziehungsweise
<b>ca.</b>	circa
<b>CG</b>	ClaimsGuard
<b>CI/CD</b>	Continuous Integration/Continuous Delivery
<b>CSS</b>	Cascading Style Sheets
<b>DI</b>	Dependency Injection
<b>DBMS</b>	Datenbankmanagementsystem
<b>DTO</b>	Data Transfer Object
<b>etc.</b>	et cetera
<b>ggf.</b>	gegebenenfalls
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>Icam</b>	IcamSystems GmbH
<b>IDE</b>	Integrated Development Environment
<b>IoC</b>	Inversion of Control
<b>JSON</b>	JavaScript Object Notation
<b>JS</b>	JavaScript
<b>MVC</b>	Model View Controller
<b>PDF</b>	Portable Document Format
<b>SME</b>	Subject Matter Expert
<b>SPA</b>	Single-Page Application
<b>u.a.</b>	unter anderem
<b>VID</b>	Vehicle Information Database
<b>XML</b>	Extensible Markup Language
<b>z.B.</b>	zum Beispiel

# 1 Einleitung

## 1.1 Projektumfeld

Die IcamSystems GmbH (Icam) bietet Softwarelösungen im Bereich der Kfz-Schadensregulierung an. Sie ist Teil eines **Firmenverbundes** mit ca. 160 Mitarbeitern. Zu den Kunden zählen unter anderem Versicherungsunternehmen, Sachverständigenorganisationen, Prüfdienstleister und Schadensteuerungsgesellschaften.

Zum Leistungsspektrum gehört die automatisierte Prüfung von Gutachten und Kostenvoranschlägen, welche durch Gutachter oder Werkstätten erstellt wurden.

Das Projekt wurde intern durch das Team Reklamationsbearbeitung aus der Abteilung Prozessautomatisierung in Auftrag gegeben. Die Fachabteilung setzt unternehmensspezifische Prozesse u.a. mittels Business Process Management Software (BPMS) um.

## 1.2 Projektziel

Ziel ist eine Webanwendung mit einer funktionalen Benutzeroberfläche, welche Daten von einem Backend-Service abfragt, um diese strukturiert und übersichtlich darzustellen. Der Prozess der Reklamationsbearbeitung soll so schneller bearbeitet werden können und die händische Suche nach allen Teilinformationen überflüssig machen.

## 1.3 Projektbegründung

Die Vorgangsprüfung erfolgt über das firmeneigene, automatisierte Prüfgewerk ClaimsGuard (CG). Darin prüft ein individuell erstelltes Regelwerk die, z.B. von Werkstätten, erstellten Kostenvoranschläge und Gutachten. Für die individuellen Parameter jedes Fahrzeugtyps greift der CG auf die recherchierten Fahrzeugdaten der Vehicle Information Database (VID) zurück. Die VID bietet für jedes Kfz-Bauteil Informationen zur Beschaffenheit, Verarbeitung und Lage im Fahrzeug. Dort sind auch die Daten zu den Fahrzeugmodellen insgesamt abgelegt.

Bei der Plausibilitätsprüfung durch den CG entsteht ein detaillierter Prüfbericht, in dem Diskrepanzen aufgeführt werden.

Sollte der Kunde das Prüfergebnis beanstanden, so hat er die Möglichkeit eine Reklamation an die Icam zu senden.

Aktuell können nur die Projektverantwortlichen selbst, mittels des jeweiligen Aktenzeichens der Reklamation, eine händische Suche in mehreren Datenbanken und dem Regelwerk des CG durchführen, um den Vorgang nachzuvollziehen. Es soll für die Bewertung des Vorgangs kompakt ersichtlich sein, welche Regeln ausgelöst wurden und welche Daten der VID in die Kalkulation einbezogen wurden.

Dieser Prozess ist aufgrund des hohen Anteils manueller Arbeit zeitaufwändig und fehleranfällig.



Weiterhin kann dieser, wegen der Komplexität der Datenbanksysteme, nur von Experten durchgeführt werden. **Durch** das ReklaTool sollen die Suchanfragen vereinfacht und standardisiert werden, was wiederum eine Erweiterung des Nutzerkreises möglich macht.

## **1.4 Projektschnittstellen**

### **Technische Schnittstellen**

In der Abteilung Prozessautomatisierung wurde mittels BPMS eine Datenbank-API erstellt. Diese trägt Daten aus Produktiv- und Archivdatenbanken zusammen. Die Daten werden mit den fahrzeugspezifischen Daten aus der VID ergänzt. Hinzu kommen die im Vorgang ausgelösten Regeln des CG. Dieses Datenpaket wird der Webanwendung in strukturierter Form zur Verfügung gestellt.

### **Verantwortlichkeit**

**Das Projekt** wird durch die Abteilung Projektmanagement begleitet.

### **Benutzer der Anwendung**

Anwender sind zum jetzigen Stand die Projektverantwortlichen des CG. Aufgrund der bisherigen Komplexität der Datenbankabfragen waren bisher nur Subject Matter Expert (SME) für die Abfragen zuständig. Durch die Vereinfachung dieses Vorgangs, ist eine Erweiterung des Nutzerkreises denkbar.

### **Endabnahme**

Die technische Abnahme und die Auslieferung wird von der IT-Abteilung umgesetzt. Das Ergebnis des Projekts wird abschließend von den SME der Fachabteilung getestet und abgenommen. Die Dokumentation wird der Projektmanagementabteilung übergeben.

## **1.5 Projektabgrenzung**

Die API für den Datenbankzugriff wurde via BPMS in der Abteilung Prozessautomatisierung realisiert und bereitgestellt. Sie existiert unabhängig vom ReklaTool.

## 2 Projektplanung

### 2.1 Projektphasen

Für die Bearbeitung des Projekts standen dem Autor im Projektzeitraum täglich etwa 5 Stunden zur Verfügung. Insgesamt wurde das Projekt in 80 Stunden umgesetzt. Diese Zeit wurde in Phasen aufgeteilt, welche den Projektablauf widerspiegeln. Aus Tabelle 1 kann die grobe Zeitplanung entnommen werden.

Projektphase	Geplante Zeit
Analyse	8 h
Entwurf	16 h
Implementierung und Tests	40 h
Abnahme	3 h
Dokumentation	13 h
<b>Gesamt</b>	<b>80 h</b>

Tabelle 1: Zeitplanung

Eine detailliertere Zeitplanung befindet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

### 2.2 Ressourcenplanung

Im Anhang A.2: Ressourcen auf Seite ii befindet sich eine Auflistung der verwendeten Ressourcen. Dort werden sowohl Hardware- und Softwareressourcen, als auch das beteiligte Personal aufgeführt. Es wurde nur auf Software zurückgegriffen, für die bereits Lizenzen im Unternehmen vorhanden waren, bzw. die kostenfrei genutzt werden konnte. Dieser Umstand wirkte sich positiv auf die Projektkosten aus.

### 2.3 Entwicklungsprozess

Das Projekt wurde vom Autor als einzelner Entwickler in einem überschaubaren Zeitraum von drei Wochen umgesetzt. Aus diesen Gründen und da die Anforderungen zu Beginn schon klar definiert wurden, wurde das Projekt anhand der Phasen des Wasserfallmodells umgesetzt. Eine Eigenschaft des Wasserfallmodells ist die lineare Abfolge der einzelnen Projektphasen. Bei der Implementierung wurde bewusst ein inkrementeller Ansatz gewählt, um die Produktqualität zu gewährleisten.

Besonders bei Analyse und Entwurf war die Rücksprache mit dem Fachbereich vorgesehen. Zur Sicherstellung der späteren Wartbarkeit wurde die Webanwendung nach den fünf SOLID-Prinzipien für objektorientierte Programmierung entworfen.



## 3 Analysephase

Um die Anforderungen an das ReklaTool zu bestimmen, wurde der aktuelle Prozess zur Reklamationsbearbeitung zusammen mit der Fachabteilung erfasst. Aus der Analyse des Vorgangs ergaben sich einzelne Anwendungsfälle (Use-Cases) für die zu erstellende Webanwendung. Basierend auf dem Umfang der Anwendung wurde eine Wirtschaftlichkeitsanalyse durchgeführt und schließlich das Lastenheft erstellt.

### 3.1 Ist-Analyse

Um den im Folgenden beschriebenen Prozess zu verbildlichen, befindet sich ein Schaubild im Anhang A.5: Aktivitätsdiagramm auf Seite iv.

Zum Leistungsspektrum der Icam gehört die automatisierte Prüfung von Gutachten und Kostenvoranschlägen, welche durch Gutachter oder Werkstätten erstellt wurden. Die Prüfung wird durch die Versicherungen oder Sachverständigenorganisationen (Kunden) veranlasst und wird durch den CG durchgeführt. Als Ergebnis bekommt der Kunde zusätzlich zum Prüfbericht eine Rückmeldung über die ausgelösten Regeln des CG.

Im Fall, dass die Regelauslösungen für den Kunden nicht nachvollziehbar sind, hat er die Möglichkeit eine Reklamation an die Icam zu senden. Diese wird firmenintern geprüft und an die Fachabteilung Prozessautomatisierung weitergeleitet.

Ein Mitarbeiter des Teams Reklamationsbearbeitung recherchiert anschließend mittels des Aktenzeichens des Vorgangs die dazugehörigen Daten. Je nach Zeitpunkt des Vorgangs können die Daten in Produktiv- oder Archivdatenbanken abgelegt sein. Der Mitarbeiter sucht mittels Datenbankmanagementsystem (DBMS) und selbst erstellten SQL-Skripten nach allen Daten zum Vorgang.

Mit diesen Daten kann dann die beanstandete Regelauslösung nachvollzogen werden. Dazu bietet der CG die Möglichkeit die Regelauslösungen des Vorgangs Schritt für Schritt durchzugehen. Fällt dabei eine fehlerhafte Prüfregel auf, so wird diese angepasst.

Liegt die Ursache der fehlerhaften Regelauslösung nicht in der Regelimplementierung, wird die Reklamation an die Rechercheabteilung gegeben. Diese prüft die Fahrzeugdaten des Vorgangs auf Vollständigkeit und Korrektheit. Auch hier werden bei Auffälligkeiten Daten korrigiert oder nachrecherchiert und ergänzt.

Der Kunde bekommt Rückmeldung über die angepassten Daten und Prüfregeln. Sind die Daten jedoch nach Prüfung initial richtig gewesen, so bekommt der Kunde auch darüber eine Rückmeldung, inklusive von Quellen (z.B. Herstellerdaten) als Nachweis. Danach ist der Prozess abgeschlossen.

## 3.2 Wirtschaftlichkeitsanalyse

Wie bereits im Abschnitt 3.1: Ist-Analyse zu erkennen ist, war der vorherige Reklamationsprozess mit viel manueller Recherchearbeit verbunden. Im Folgenden wird analysiert, ob das Projekt durch die Zeitersparnis, welche mit dem ReklaTool einhergeht, wirtschaftlich sinnvoll ist.

### 3.2.1 „Make or Buy“-Entscheidung

Das ReklaTool greift mittels der Datenbank-API auf sensible Firmendaten zu, mit denen auch Verpflichtungen gegenüber Kunden **einhergehen**. Weiterhin soll die Webanwendung unternehmensspezifischen Anforderungen genügen. So liegen Daten zwar im XML-Format vor, sind aber je nach Anforderung stark firmenspezifisch strukturiert. Aus diesen Gründen ist von dem Beziehen von Software von Dritten abzusehen und die Eigenentwicklung vorzuziehen.

### 3.2.2 Projektkosten

Dieser Abschnitt betrachtet die Kosten, welche bei der Umsetzung des Projekts entstehen.

Diese setzen **aus** Personalkosten und aus Kosten für die verwendeten Ressourcen (siehe Kapitel 2.2: Ressourcenplanung) zusammen und können der Tabelle 2 entnommen werden.

Die Personalkosten (brutto) je Projektmitarbeiter wurden durch die Personalabteilung vorgegeben. Für einen Mitarbeiter der IT-Abteilung wurde ein Stundensatz in Höhe von 30,00 € angenommen. Mitarbeiter der Fachabteilung gehen mit 35,00 € pro Stunde in die Berechnung ein. Da der Autor aufgrund seiner Umschulung nicht direkt von der Praktikumsfirma bezahlt wird, wird für diesen das Azubigehalt des dritten Lehrjahrs, mit 7,50 € pro Stunde angenommen.

Die Betriebskosten für die Webanwendung werden mit jährlich 110,00 € beziffert und für die Nutzung der Ressourcen gilt ein Pauschalbetrag von 20,00 € pro Stunde.

Die Gesamtkosten des Projekts betragen somit 2.575,00 €.

Vorgang	Beteiligung	Zeit	Kosten pro Stunde	Kosten
Entwicklung	Autor	80 h	7,50 € + 20,00 € = 27,50 €	2.200,00 €
Code-Review	IT-Abteilung	1 h	30,00 € + 20,00 € = 50,00 €	50,00 €
Abnahme technisch	IT-Abteilung	1 h	30,00 € + 20,00 € = 50,00 €	50,00 €
Analyse/Lastenheft/Mock-Ups	Fachabteilung	4 h	35,00 € + 20,00 € = 55,00 €	220,00 €
Abnahme fachlich	Fachabteilung	1 h	35,00 € + 20,00 € = 55,00 €	55,00 €
				<b>2.575,00 €</b>

Tabelle 2: Kostenaufstellung

### 3.2.3 Amortisation

Der Wegfall der in Abschnitt 3.1: Ist-Analyse beschriebenen Recherchearbeit in mehreren Datenbanken schlägt sich in einer Zeitersparnis nieder, welche sich u. a. über die Lohnkosten als finanzieller Vorteil beziffern lässt.

Vorgang	Anzahl pro Jahr	Zeit (alt) pro Vorgang	Zeit (neu) pro Vorgang	Ersparnis pro Jahr
Entwicklung	110	15 min	1 min	1.540 min
				<b>1.540 min</b>

Tabelle 3: Zeitersparnis

### Berechnung der Amortisationsdauer

Die Ermittlung der Amortisationsdauer soll dabei helfen, die Wirtschaftlichkeit des Projekts zu beurteilen. Für die Berechnung wird die vorher im Abschnitt 3.2.2: Projektkosten kalkulierte Gesamtsumme mit den Einsparungen verglichen. Zur Ermittlung der finanziellen Ersparnis wird die jedes Jahr gewonnene Zeit mit dem Stundensatz, plus der Ressourcenpauschale, multipliziert. Es ergeben sich somit:

$$\frac{1.540 \text{ min/Jahr}}{60} \cdot (35 + 20) \text{ €/h} \approx 1.411,67 \text{ €/Jahr.} \quad (1)$$

Um nun die Amortisationsdauer zu berechnen, werden die Projektkosten mit den jährlichen Ersparnissen ins Verhältnis gesetzt. Dabei werden auch die Betriebskosten des ReklaTool von 110,00 € pro Jahr mit beachtet.

$$\frac{2.575,00 \text{ €}}{(1.411,67 - 110,00) \text{ €/Jahr}} \approx 2 \text{ Jahre.} \quad (2)$$

In der Abbildung Anhang A.3: Break-Even-Point auf Seite iii ist die Amortisierung grafisch dargestellt. An der Stelle, wo sich Projektkosten und Einsparung nach einer bestimmten Zeit treffen, lässt sich der *Break-Even-Point* ablesen. Durch diesen hat der Autor eine Aussage darüber, ab welchem Zeitpunkt sich das Projekt amortisiert hat.

Da sich der Arbeitsablauf der Fachabteilung auf absehbare Zeit wenig verändern wird, ist die Amortisierung nach knapp zwei Jahren als wirtschaftlich zu betrachten. Hinzu kommen die im folgenden Kapitel beschriebenen Aspekte.

### **3.3 Nicht-monetärer Nutzen**

Der alte Reklamationsprozess (Vgl. Kapitel 3.1: Ist-Analyse) wurde aufgrund seiner Komplexität meist von SME durchgeführt. Durch die hohe Priorität der Reklamationen wurde der Arbeitsablauf dieser Mitarbeiter für einen längeren Zeitraum unterbrochen. Diese Unterbrechung wird durch den automatisierten Zugriff auf alle relevanten Quellen mit dem ReklaTool verkürzt. Durch die einfachere Handhabung ist nun auch die Einbeziehung weiterer Mitarbeiter in das Reklamationsmanagement möglich.

Weiterhin wird durch die Vereinheitlichung des Prozesses und die Reduzierung manueller Teilschritte, die Fehlerquote minimiert.

### **3.4 Anwendungsfälle**

Zusammen mit der Fachabteilung wurden während der Analyse Anforderungen definiert und daraus Anwendungsfälle abgeleitet. Im Anhang A.6: Use-Case-Diagramm auf Seite v befindet sich das dabei entstandene Use-Case-Diagramm.

Die Ausdifferenzierung der einzelnen Fälle diente später im Entwurf als Richtlinie für einzelne Features.

### **3.5 Qualitätsanforderungen**

Die Software soll eine funktionale und übersichtliche Benutzeroberfläche bieten. Diese soll ohne Installation, in Form einer Webanwendung im Browser erreichbar sein. Nach einer Nutzeranfrage an die Anwendung soll das Ergebnis in einer festgelegten Zeit erscheinen, um den Arbeitsablauf nicht zu sehr zu stören. Die Funktionalität und Richtigkeit der Software soll mittels Tests sichergestellt werden.

### **3.6 Lastenheft**

Aus den zusammengetragenen Anforderungen erstellte der Autor mit der Fachabteilung das Lastenheft. Darin sind alle gewünschten Funktionen und Eigenschaften der zu erstellenden Software festgeschrieben.

Ein Auszug befindet sich im Anhang A.4: Lastenheft (Auszug) auf Seite iii.

## 4 Entwurfsphase

Dieser Abschnitt behandelt alle wichtigen Entscheidungen bezüglich der Grundprinzipien, Techniken und Technologien zum Erstellen der Webanwendung.

### 4.1 Zielplattform

Wie unter 1.2: Projektziel erwähnt, soll eine Webanwendung erstellt werden. Aus der IT-Abteilung kamen dazu Informationen zu bereits in anderen Projekten eingesetzten Technologien.

Neue Software wird dort überwiegend in der Programmiersprache C# umgesetzt. Dadurch existiert für diese bereits umfangreiches Wissen bei den Mitarbeitern. Bereits vorhandene Ressourcen und Lizenzen konnten für das Projekt genutzt werden. Um das Endprodukt besser im Team wartbar und erweiterbar zu halten, hat sich der Autor ebenfalls für C# in der .NET-Umgebung von Microsoft entschieden.

Unter weiterer Rücksprache mit der Softwareabteilung wurden verschiedene Ansätze zur Umsetzung einer Webanwendung durchgesprochen. Besonders wurde hier zwischen client- und serverbasierten Anwendungen unterschieden. Da die Benutzerinteraktion eher gering ausfallen würde, wurde sich für die Servervariante entschieden. Diese ist gegenüber einer Single-Page Application (SPA) auf einem Client weniger agil und die Reaktionszeit ist eingeschränkt. Durch Nutzung des *TelerikUI*-Framework und der damit verbundenen Bindung der Daten an die Komponenten der Oberfläche entsteht eine Art Mischform der zuvor genannten Varianten.

Die Wahl des Webframework fiel auf *ASP.Net Core MVC*, zu dem es schon gute Erfahrungen im Team gab. Es bietet bereits Projektvorlagen mit der grundlegenden Programmstruktur und Möglichkeiten zur Konfiguration an.

### 4.2 Architekturdesign

Wie bereits an der Wahl des Webframeworks erkennbar ist, basiert die Anwendung auf dem MVC-Entwurfsmuster. Dieses beinhaltet die Teile **Model**, **View** und **Controller** – also Datenabbildung, Benutzeroberfläche und Vermittlung. Durch die klare Trennung in diese drei Bereiche vermindern sich die Abhängigkeiten untereinander. Implementierungen dieser Teile können leichter verändert oder sogar ausgetauscht werden. Ein weiterer Vorteil ist der gut nachvollziehbare Datenfluss innerhalb der Anwendung.

Dieser Kern des Programms wird durch Services erweitert, welche jeweils eine spezielle Aufgabe haben. Dazu bietet das Framework einen Injektor, mit dem das Prinzip der *Inversion of Control* via *Dependency Injection* umgesetzt wird. Dabei müssen sich die Teile des Programms nicht um die Erstellung von Instanzen ihrer Abhängigkeiten kümmern, sondern Bekommen diese vom Injektor übergeben. Dieser übernimmt das komplette Management der erstellten Objekte und kann diese, je nach Lebensdauer, ggf. entfernen, sollten sie nicht mehr gebraucht werden.

Die daraus folgende lose Verbindung der einzelnen Module trägt weiter zur besseren Wartbarkeit

der Anwendung bei.

Zur Steuerung der Kommunikation mit der Datenbank-API sollte ein eigener Service entwickelt werden. Dieser übermittelt Anfragen und stellt die Antworten der Anwendung zur Verfügung. Um die Performance von mehrfach aufgerufenen Vorgängen zu erhöhen, wird **dieser** um einen *Caching*-Service erweitert. Beim *Caching* werden die erhaltenen Daten in Dateien gespeichert, welche bestimmten Abfragen zugeordnet und später wieder eingelesen werden können. Das *Model* wird mittels verhaltensloser Datenklassen (DTOs) erstellt.

### 4.3 Entwurf der Benutzeroberfläche

Um den einfachen Zugriff von beliebigen Rechnern im Firmenintranet zu gewährleisten, fiel die Entscheidung auf eine Umsetzung mit Weboberfläche. Diese soll in dem vom Unternehmen genutzten Browser angezeigt werden können. Da die Anwendung lediglich auf Desktopcomputern zum Einsatz kommen wird, wurde bewusst auf Anpassungen für mobile Geräte verzichtet **und** der Fokus auf eine funktionale Gestaltung gelegt.

Über mehrere Korrekturschleifen wurde zusammen mit der Fachabteilung der Entwurf für eine Benutzeroberfläche fertiggestellt. Mithilfe von *Mockups* wurden die Bedürfnisse der Endbenutzer so gut wie möglich nachempfunden. Diese befinden sich im Anhang A.14: Oberflächenentwurf auf Seite xii.

Die Webseite soll in der Hauptsache aus einer Suchleiste für Nutzeranfragen und einem Bereich zur Darstellung der Vorgangsdaten bestehen. Eine Checkbox soll die Möglichkeit geben, den Umfang der Suche einzuschränken.

Die Vorgangsdaten werden nach erfolgreicher Suche in einem Bereich angezeigt, der die einzelnen Daten mittels Registerkarten in Kategorien unterteilt. Zur strukturierten Darstellung werden filterbare Tabellen genutzt. Diese werden durch das Framework *TelerikUI* eines Drittanbieters bereitgestellt und bieten vielfältige Einstellungen zum Filtern, Sortieren und Gruppieren an. Eine Lizenz ist hierfür bereits im Unternehmen vorhanden. Weiterhin soll es möglich sein, eine PDF- oder XML-Datei mit den Vorgangsdaten anzuzeigen bzw. diese zum Download anzubieten (siehe 3.4: Anwendungsfälle).

### 4.4 Datenmodell

Die Datenmodelle für sowohl Anfragen an die Datenbank-API, als auch deren Antwort basieren auf den versendeten XML-Dateien. Diese wurden vom Fachbereich bereitgestellt und durch Deserialisierung in DTO-Klassen überführt.

Im Anhang A.12: Datenmodell auf Seite x befindet sich dazu eine Übersicht.

## 4.5 Geschäftslogik

Unter den Anwendungsfällen ist das Abfragen der Daten von der Datenbank-API der wichtigste. Anhand diesem wird ein Großteil des Programmablaufs nachvollziehbar und kann im Sequenzdiagramm im Anhang 6: Sequenzdiagramm auf Seite vii betrachtet werden.

Im gezeigten Fall ruft der Nutzer über seinen Browser das *ReklaTool* auf. Dort gibt er über das Suchfeld das Aktenzeichen zum gesuchten Vorgang ein. Da die Aktenzeichen kein festes Format besitzen, muss zusätzlich deren Art (Dateiname, Vorgangsnummer, Schadennummer oder Auftragsnummer) ausgewählt werden. Um die Suche einzuschränken, kann ein Haken gesetzt werden, ob Dokumente zum Vorgang mitgeliefert werden sollen. Nach Betätigung der Suchen-Taste wird eine Anfrage mit Aktenzeichen und Filter an die *ReklaTool*-API gestellt. Dort bietet der *VorgangController* Endpunktmethoden für HTTP-Anfragen an. Über den *UiEndpointService* werden die Anfragen an den *HttpMsgService* durchgereicht. Dieser erstellt mittels des *Request-Builder* ein Anfrage-Objekt.

Die erzeugte Anfrage wird zunächst durch den *ResponseCacheService* mit den bereits zwischengespeicherten Vorgängen verglichen. Ist eine Datei zum Aktenzeichen vorhanden, so wird diese an den *UiEndpointService* zurückgegeben. Andernfalls wird nun eine Anfrage an die Datenbank-API gestellt. Das zurückgelieferte XML-Dokument wird deserialisiert und zusammen mit den Anfrageinformationen im *Cache* abgelegt. Danach geht auch dieses zurück an den *UiEndpointService*. Hier werden die Daten, je nach Client-Anfrage, vom kompletten Vorgangs-Model auf einzelne *View-Models* gemappt. Zurück im *VorgangController* wird nun die Server-Response in Form einer JSON-Datei zurück an den Client gesendet und die Daten an die entsprechenden Komponenten gebunden, wo sie für den Nutzer sichtbar werden.

## 4.6 Maßnahmen zur Qualitätssicherung

Um die einzelnen Schritte während der Implementierung nachvollziehbar zu halten, wird das Projekt in der Quellcodeverwaltung *Git* abgelegt. Zur Vereinfachten Handhabung von *Git* wird im Unternehmen auf die Software *SmartGit* gesetzt. Diese zeigt Änderungen im Projekt an und beherrscht alle typischen Funktionen und Workflows im Zusammenhang mit der Versionskontrolle. Bei Fertigstellung einer Implementierungseinheit wird das lokale Projekt in das Projekt-*Repository* im Firmennetz gepusht. Zur Verwaltung der Projekte setzt die Icam hierbei auf GitLab als on-premises Installation. Zum Zweck des Codereview werden Mitarbeiter der IT-Abteilung zum GitLab-Projekt hinzugefügt.

Die Qualitätssicherung während der Entwicklung der Webanwendung wurde ursprünglich durch Unit-Tests geplant (siehe Seite : Projektantrag). Da die einzelnen Services nur auf Basisfunktionen des Frameworks zurückgreifen und die Komplexität des Programms überwiegend durch die Verbindung der Module untereinander entsteht, hat sich der Autor auf die Durchführung von Integrationstests beschränkt. Diese werden in Form von *Whitebox-Tests* umgesetzt.

## 4.7 Pflichtenheft

Aus dem Lastenheft (Anhang A.4: Lastenheft (Auszug) auf Seite iii) heraus wurde das Pflichtenheft erstellt. Dabei hat der Autor wichtige Implementierungsdetails ergänzt, welche ebenfalls mit der Fachabteilung abgestimmt **worden sind**. Ein Auszug ist im Anhang A.10: Pflichtenheft (Auszug) auf Seite viii einsehbar. Es dient zur Kontrolle, ob alle Anforderungen planmäßig umgesetzt wurden.



## 5 Implementierungsphase

Bei der Implementierung wurden die in der Entwurfsphase beschriebenen Architekturvorgaben und Programmteile umgesetzt. Begonnen wurde mit dem Erstellen des Projekts nach einer vom *Telerik-Framework* bereitgestellten Vorlage für eine *ASP.Net Core MVC*-Anwendung. Diese liefert bereits *Boilerplate-Code*, der z. B. die Konfiguration vereinfachen soll.

### 5.1 Konfiguration

Das *ASP.Net Core MVC*-Framework bietet verschiedene Möglichkeiten zur Konfiguration. So konnten über die Datei `appsettings.json` Information wie API-Adresse und Daten für die Authentifizierung in die Anwendung gegeben werden. Diese wurden durch das Interface `IConfiguration` via DI den anderen Services zur Verfügung gestellt.

In der Datei `Program.cs` werden alle Services mit entsprechender Lebensdauer (scoped, transient oder singleton) registriert.<sup>1</sup> Für das Projekt wurden diese als *scoped* angemeldet. Das bedeutet, dass der Service immer für die Dauer einer API-Anfrage zur Verfügung steht. Eine Beschreibung, wie Services in das Programm eingebunden sind, findet sich im Anhang A.18: Entwicklerdokumentation (Auszug) auf Seite xvii.

Nach der Registrierung der Dienste wurden in der `Program.cs` verschiedene *Middleware*-Anwendungen registriert und konfiguriert. Diese können vor bzw. nach dem Programmcode die Anfrage oder die Antwort auswerten und bearbeiten. Ein Typisches Beispiel ist die Autorisierung eines Benutzers.

### 5.2 Authentifizierung

Für die Authentifizierung und Autorisierung setzt die Icam auf den Standard *OAuth2*. Zur Umsetzung wurde dabei auf die Software **Keycloak** zurückgegriffen. Dazu hat die IT-Abteilung einen Client angelegt. Die Zugangsdaten mussten dann nur der Anwendung mitgeteilt werden. Durch diverse Softwarebibliotheken wurde der Authentifizierungsprozess in die *Middleware* von ASP.NET eingefügt. Durch das Attribut `[Authorize]` wurde der `VorgangController` vor unberechtigtem Zugriff geschützt. Die Verwendung des Keycloak und OAuth2 sorgte für eine schnellere Entwicklung, da die Benutzerverwaltung nicht mehr implementiert werden musste. Diese wird extern von der IT-Abteilung über das, an den Keycloak angeschlossene *Active Directory* verwaltet. Durch diesen Vorgang wurde die Implementierung der Nutzerauthentifizierung verkürzt (siehe dazu 9.1).

---

<sup>1</sup><https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection#service-lifetimes>

### 5.3 Implementierung der Datenstrukturen

Zur Erstellung der einzelnen DTOs wurden die XML-Dateien deserialisiert, über die der Informationsaustausch mit der Datenbank-API realisiert wird. Dabei handelte es sich um jeweils ein Anfrage- und ein Antwortformat. Diese wurden von der Fachabteilung zur Verfügung gestellt. Das Anfrage-Model beinhaltet das jeweils gewünschte Aktenzeichen und eine Information über die Anwendung eines Suchfilters, welcher die zurückgelieferten Daten einschränkt.

Das Antwort-Model wurde bei der Deserialisierung bereits automatisch mit den notwendigen Annotationen versehen, um einen reibungslosen Austausch mit dem XML-Format zu garantieren. Zur Datenbindung an die einzelnen Komponenten der Benutzeroberfläche wurden weitere Viewmodels erstellt, welche das Mapping auf das Antwort-Model ermöglichen.

Für das Caching wurde ein eigenes DTO umgesetzt, welches das Anfrage- und Antwort-Model enthält. Dieses wird beim Lesen und Schreiben der Cache-Dateien serialisiert bzw. deserialisiert. Über das Hashing des Anfrage-Objekts sind die Dateien eindeutig identifizierbar.

### 5.4 Implementierung der Benutzeroberfläche

Bereits in 4.3: Entwurf der Benutzeroberfläche wurde festgelegt, dass die Benutzeroberfläche als Webseite in einem Browser ausgeführt werden soll.

Die von TelerikUI bereitgestellte Vorlage für eine Model View Controller (MVC)-Anwendung verlangt die Implementierung der View mittels spezieller .cshtml-Dateien. Diese erlauben, mittels definierter Schlüsselwörter, das Verwenden von C#-Syntax in einem html-Dokument. Beides zusammen wird innerhalb des ASP.NET-Ökosystems als Razor-Syntax bezeichnet.<sup>2</sup>

Die Oberflächenkomponenten, z. B. Buttons, Suchfelder oder Tabellen, werden daher auch in C# definiert und konfiguriert. Siehe dazu Anhang A.15: View (Auszug) auf Seite xii.

Um die ereignisbasierte Interaktion mit dem Nutzer zu gestalten, wurde, wie bei den meisten Webseiten, auf die Verwendung der Programmiersprache JS im Zusammenspiel mit Ajax zurückgegriffen.<sup>3</sup>

### 5.5 Implementierung der Geschäftslogik

Laut 3.4 Anwendungsfälle ist der Hauptanwendungsfall das Abfragen der Daten der Datenbank-API. Anhand dessen wurde in der Entwurfsphase ein Sequenz-Diagramm erstellt (Vgl. Anhang 6: Sequenzdiagramm auf Seite vii).

Nachdem das Projekt initialisiert wurde, wurden die aus den Diagrammen ersichtlichen Serviceklassen und die dazugehörigen Interfaces implementiert. Danach konnten diese in der Datei Program.cs über den Aufruf AddScoped der builder.Services-Instanz dem IoC-Container hinzugefügt werden und standen somit dem Injektor des Frameworks zur Verfügung.

---

<sup>2</sup><https://learn.microsoft.com/de-de/aspnet/core/mvc/views/razor?view=aspnetcore-6.0>

<sup>3</sup>[https://de.wikipedia.org/wiki/Ajax\\_\(Programmierung\)](https://de.wikipedia.org/wiki/Ajax_(Programmierung))

Nach diesem Schritt wurden die Services wie im Klassendiagramm Anhang 7: Klassendiagramm auf Seite ix miteinander verknüpft. Dieses passierte über DI, speziell Konstruktorinjektion. Die Implementierung von .NET Core sieht ein Injizieren mittels Konstruktorparameter vor. Dieses hat, gegenüber der Property-Injektion den Vorteil, dass beim Instantiieren der Klasse bereits fehlende Abhängigkeiten auffallen würden.<sup>4</sup>

## 6 Abnahmephase

Vor der Auslieferung des Programms wurde ein Code-Review durch die IT-Abteilung durchgeführt. Nachdem der Code freigegeben wurde, konnte eine Mitarbeiterin der Fachabteilung die einzelnen Funktionen des ReklaTool testen. Dieses hatte gleichzeitig den Vorteil, dass sie sich mit dem Programm vertraut machen konnte. Dieses Wissen kann an andere Benutzer weitergegeben werden. Nach der Abnahme der Software wurde diese an die IT-Abteilung übergeben.

## 7 Einführungsphase

Nach der Fertigstellung des Programms erfolgte die Übergabe an die IT-Abteilung. Von dort erfolgt die Auslieferung innerhalb des Firmenintranets. Updates werden anfangs von Hand durchgeführt und sollen später in einen Continuous Integration/Continuous Delivery (CI/CD)-Prozess überführt werden.

## 8 Dokumentation

Um den Einstieg in das Programm zu vereinfachen und dessen Aufbau zu beschreiben, wurde eine Dokumentation angefertigt. Diese besteht aus dem Benutzerhandbuch für die Endanwender und der Entwicklerdokumentation. Im Handbuch werden die einzelnen Funktionen der Webanwendung und die Gliederung der Benutzeroberfläche mit Texten und Bildern dargestellt. Dadurch soll die Anlernphase verkürzt werden.

Ein Auszug aus dem Benutzerhandbuch befindet sich im Anhang A.19: Benutzerdokumentation (Ausschnitt) auf Seite xx.

Die Entwicklerdokumentation wurde über Screenshots aus der IDE erstellt. Diese wurden nummeriert und in Textform beschrieben. Ausschnitte aus diesem Dokument sind im Anhang A.18: Entwicklerdokumentation (Auszug) auf Seite xvii zu finden.

---

<sup>4</sup>Dependency Injection in der Microsoft-Dokumentation:

<https://learn.microsoft.com/de-de/dotnet/core/extensions/dependency-injection>

## 9 Fazit

Durch die Einführung des **ReklaTool** konnten die Mitarbeiter spürbar entlastet werden. Die Verkürzung der Abfragedauer für benötigte Daten sorgt dafür, dass die täglichen Arbeitsabläufe weniger stark unterbrochen werden. In der gewonnenen Zeit können sich die Nutzer, zu denen auch Teamleiter gehören, anderen Aufgaben zuwenden. Sobald weitere Nutzer durch den vereinfachten Prozess hinzugefügt werden, wird sich dieser Effekt **noch mehr** verstärken. Nicht zu klein zu bewerten ist der finanzielle Aspekt dieser Optimierung, welche dadurch als Erfolg gewertet werden darf.

### 9.1 Soll-/Ist-Vergleich

In Tabelle 4 ist zu erkennen, dass die Zeitplanung überwiegend eingehalten werden konnte. Über das gesamte Projekt hinweg kam es in zwei Phasen zu Abweichungen von je drei Stunden. Die in 5.2 Authentifizierung beschriebene Vorgehensweise sorgte für eine Zeitersparnis. Diese gewonnene Zeit hat der Autor verwendet, um in Rücksprache mit der Fachabteilung die Benutzeroberfläche zusätzlich an Nutzerbedürfnisse anzupassen und weiter zu optimieren.

Phase	Geplant	Tatsächlich	Differenz
Analyse	8 h	8 h	
Entwurf	16 h	16 h	
Implementierung	40 h	40 h	+/- 3 h
• Implementierung der User-Authentifizierung	6 h	3 h	-3 h
• Erstellung der Weboberfläche	8 h	11 h	+3 h
Abnahme	3 h	3 h	
Dokumentation	13 h	13 h	
Gesamt	80 h	80 h	

Tabelle 4: Soll-/Ist-Vergleich

### 9.2 Lessons Learned

Die Umsetzung des Projekts brachte dem Autor viele Erfahrungen im Bereich der Projektplanung und -durchführung. Dabei nahm die Kommunikation mit den Kollegen der Fachabteilung und der IT-Abteilung eine besondere Stelle ein. Die Planung und Implementierung der Software waren dazu geeignet, sich weiter in die einzelnen Programmarchitekturen zu vertiefen. Dabei konnten wichtige Erkenntnisse sowohl über die verwendeten Frameworks, als auch Programmiersprachen gesammelt werden. Hervorzuheben sind hier die Wirkmechanismen der Kommunikation über HTTP.

### 9.3 Ausblick

Mit dem Projektabschluss wurden die Anforderungen der Fachabteilung erfüllt. Durch Veränderung der Prozesse innerhalb des Unternehmens entsteht jedoch durchaus eine Perspektive auf zukünftige Anpassungen des Reklatool, um diesem Umstand bestmöglich Rechnung zu tragen. Optimierungen im Hinblick auf die Erweiterung des Nutzerkreises sind genauso denkbar, wie die Erweiterung der Darstellung der von der Datenbank-API bereitgestellten Daten.

## A Anhang

### A.1 Detaillierte Zeitplanung

<b>Analysephase</b>	<b>8 h</b>
1. Analyse des Ist-Zustands	1 h
2. Make-Or-Buy-Analyse	1 h
3. Ermittlung von Use-Cases zusammen mit dem Fachbereich und Erstellung eines Use-Case-Diagramms	1 h
4. Soll-Konzept	2 h
5. Unterstützung der Fachabteilung bei der Erstellung des Lastenhefts	1 h
6. Wirtschaftlichkeitsbetrachtung mit Amortisationsrechnung	1 h
7. Zeit- und Ablaufplanung	1 h
<b>Entwurfsphase</b>	<b>16 h</b>
1. Planung der Projektstruktur (Versionskontrolle, Ordnerstruktur, etc.)	3 h
2. Erstellung von Mock-Ups der Weboberfläche	4 h
3. Entwurf der Systemabgrenzung, Erstellung eines Systemkontextdiagramms	2 h
4. Entwurf des Programmflusses, Erstellung eines Sequenzdiagramms	4 h
5. Entwurf der Klassen, Erstellung eines Klassendiagramms	3 h
<b>Implementierung und Tests</b>	<b>40 h</b>
1. Erstellung und Konfigurierung des C#-Projekts	3 h
2. Erstellen der Modell-Klassen	3 h
3. Erstellen des http-Service, inkl. Tests	8 h
4. Erstellen des Cache-Service, inkl. Tests	6 h
5. Erstellung und Ausführen von Integrationstests	6 h
6. Implementierung der User-Authentifizierung	6 h
7. Erstellung der Weboberfläche, inkl. Datenbindung und Interaktionen	8 h
<b>Abnahme</b>	<b>3 h</b>
1. Codereview und technische Abnahme	2 h
2. Abnahme durch den Fachbereich	1 h
<b>Dokumentation</b>	<b>13 h</b>
1. Erstellen der Nutzerdokumentation	4 h
2. Erstellen der Entwicklerdokumentation	1 h
3. Projektbewertung	1 h
4. Erstellen der Projektdokumentation	7 h
<b>Gesamt</b>	<b>80 h</b>

## A.2 Ressourcen

### Hardware

- Büroarbeitsplatz mit MS Windows Desktop-PC

### Software

- Microsoft Windows 10 Pro – Betriebssystem
- Microsoft Visual Studio Enterprise 2022 (64-bit) – v17.3.6 – Entwicklungsumgebung
- Microsoft MS Test – Framework für Unit-Tests in Visual Studio
- JetBrains ReSharper Tools v2022.1 – Erweiterung für Visual Studio für u. a. Refactoring
- Microsoft Visual Studio Code v1.72.1 – Texteditor mit Highlightfunktion und vielen Erweiterungen
- draw.io – Programm zum Erstellen von UML-Diagrammen
- syntevo SmartGit v21.1.1 – Clientprogramm zur Versionsverwaltung mit Git
- GitLab – intern gehostete Versionsverwaltung
- Progress Telerik UI – Framework für Benutzeroberflächen
- Google Chrome (64-bit) v105.0.5195.102 – Webbrowser
- MiKTeX – Distribution des LaTeX-Textsatzsystems
- LaTeX Workshop – Erweiterung für Visual Studio Code zum Editieren von LaTeX-Dokumenten
- Atlassian Confluence – Wiki-Software zur Projektdokumentation
- Atlassian Jira – Ticketsystem zur Projektverwaltung
- Postman – Tool zur Abfrage von Web-APIs
- Screenpresso – Tool zum Erstellen von Screenshots

### Personal

- Umschüler Fachinformatiker für Anwendungsentwicklung – Umsetzung des Projekts
- Entwickler der Softwareabteilung – Codereview, technische Abnahme, Deployment
- Entwicklerin der Fachabteilung – Anforderungen des ReklaTool, Unterstützung beim Entwurf der Benutzeroberfläche, Endabnahme der Anwendung

### A.3 Break-Even-Point

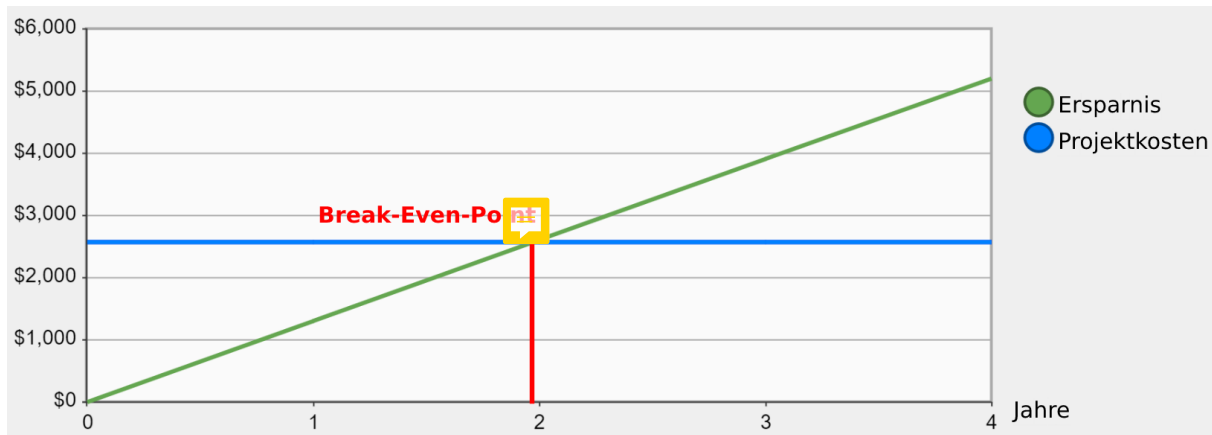


Abbildung 1: Break-Even-Point

### A.4 Lastenheft (Auszug)

Die Anwendung soll folgende Anforderungen erfüllen:

1. Abfragen der Daten
  - 1.1. Die Anwendung soll Daten zu einem kalkulierten Vorgang von einer bereitgestellten Datenbank-API abrufen.
  - 1.2. Suchanfragen sollen durch Eingabe von Aktenzeichen (Auftragsnummer, Schadennummer, Filename oder Vorgangsnummer) möglich sein.
  - 1.3. Es soll möglich sein, eine Schnellsuche durchzuführen. Dabei sollen die Daten zu Regelauslösungen und der ClaimsGuard-Prüfbericht weggelassen werden.
2. Anzeigen der Daten
  - 2.1. Auflistung aller Vorgänge zum Aktenzeichen
  - 2.2. Aufbereitung der Daten in Kategorien
  - 2.3. Darstellung von Daten mithilfe sortierbarer Tabellen
  - 2.4. Darstellung der von der API gelieferten PDF-Datei
3. Zusatzfunktionen
  - 3.1. Download der von der API mitgelieferten PDF-Datei
  - 3.2. Download der von der API mitgelieferten XML-Datei



## A.5 Aktivitätsdiagramm

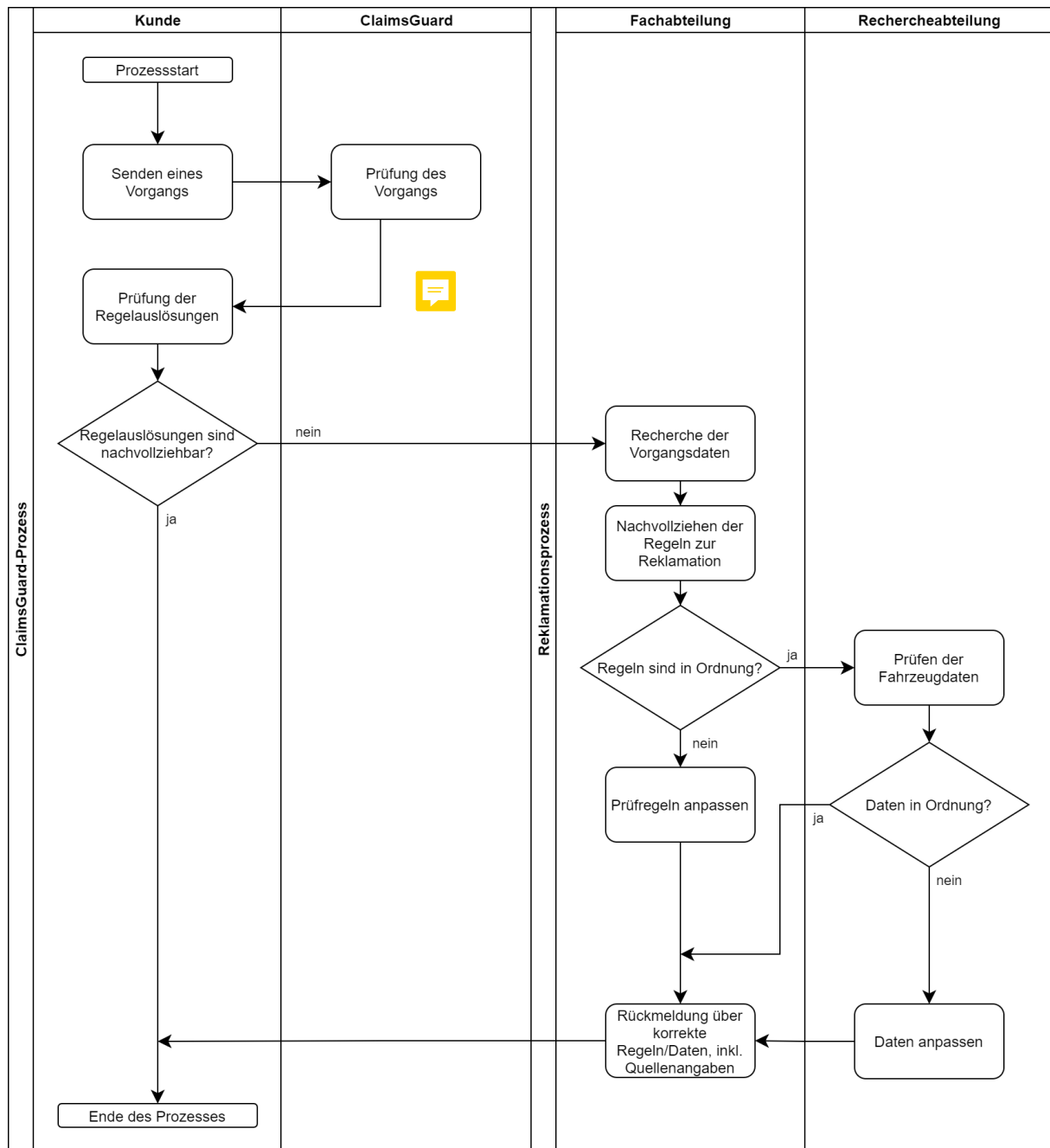


Abbildung 2: Aktivitätsdiagramm

## A.6 Use-Case-Diagramm

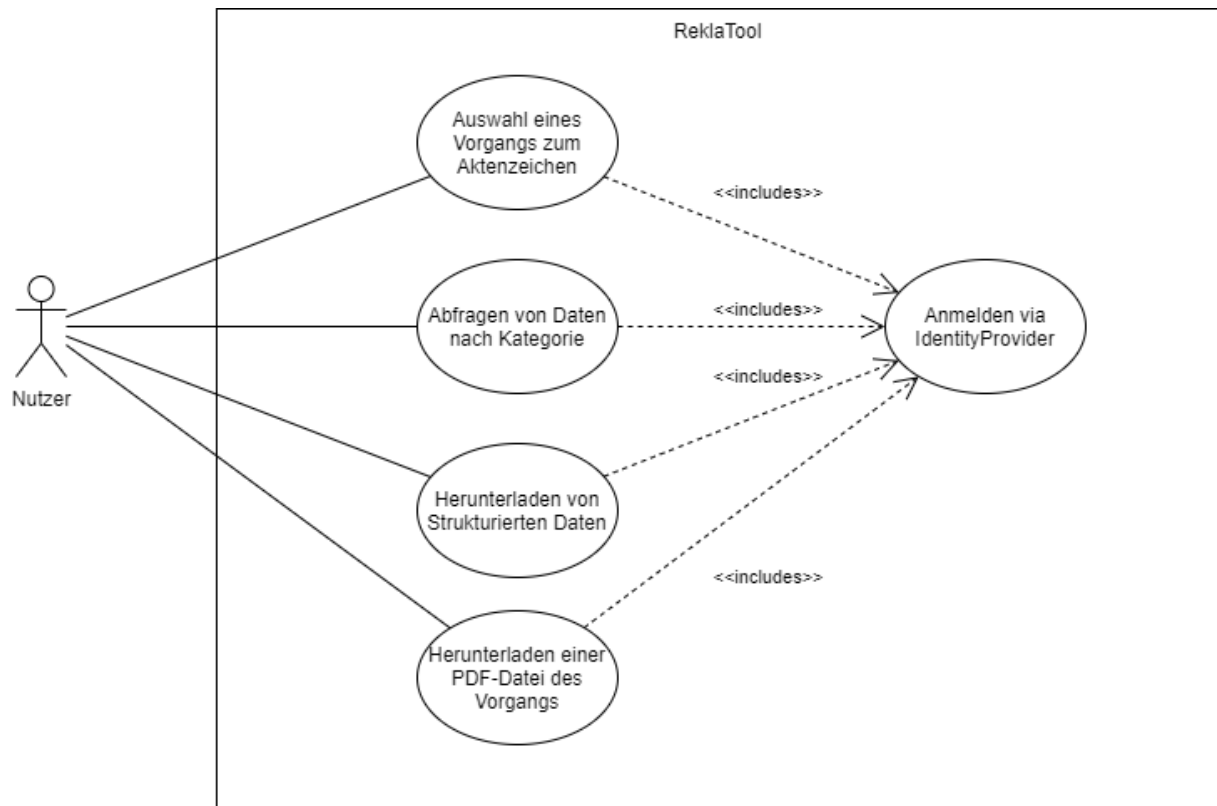


Abbildung 3: Use-Case-Diagramm

## A.7 Systemkontextdiagramm

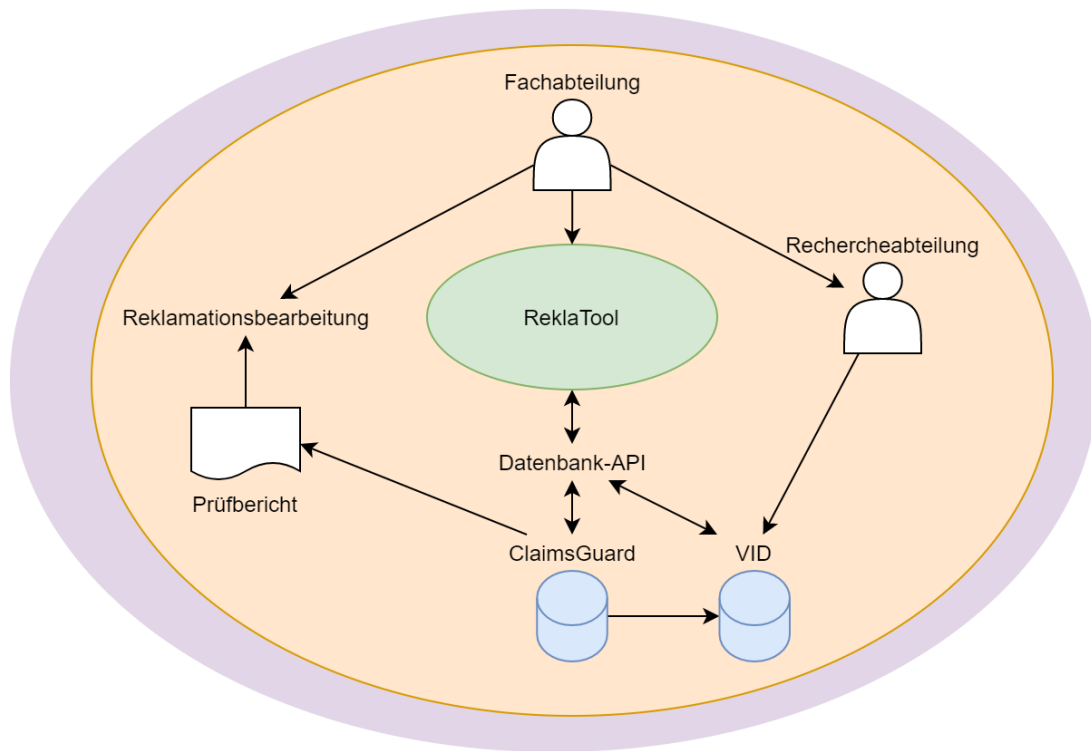


Abbildung 4: Systemkontextdiagramm

## A.8 MVC-Entwurfsmuster

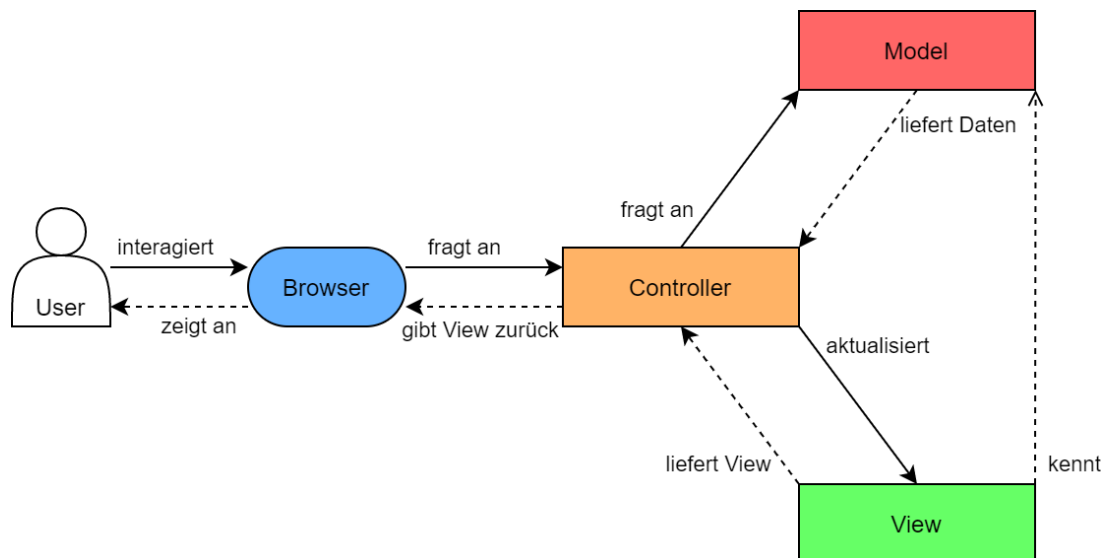


Abbildung 5: MVC-Entwurfsmuster

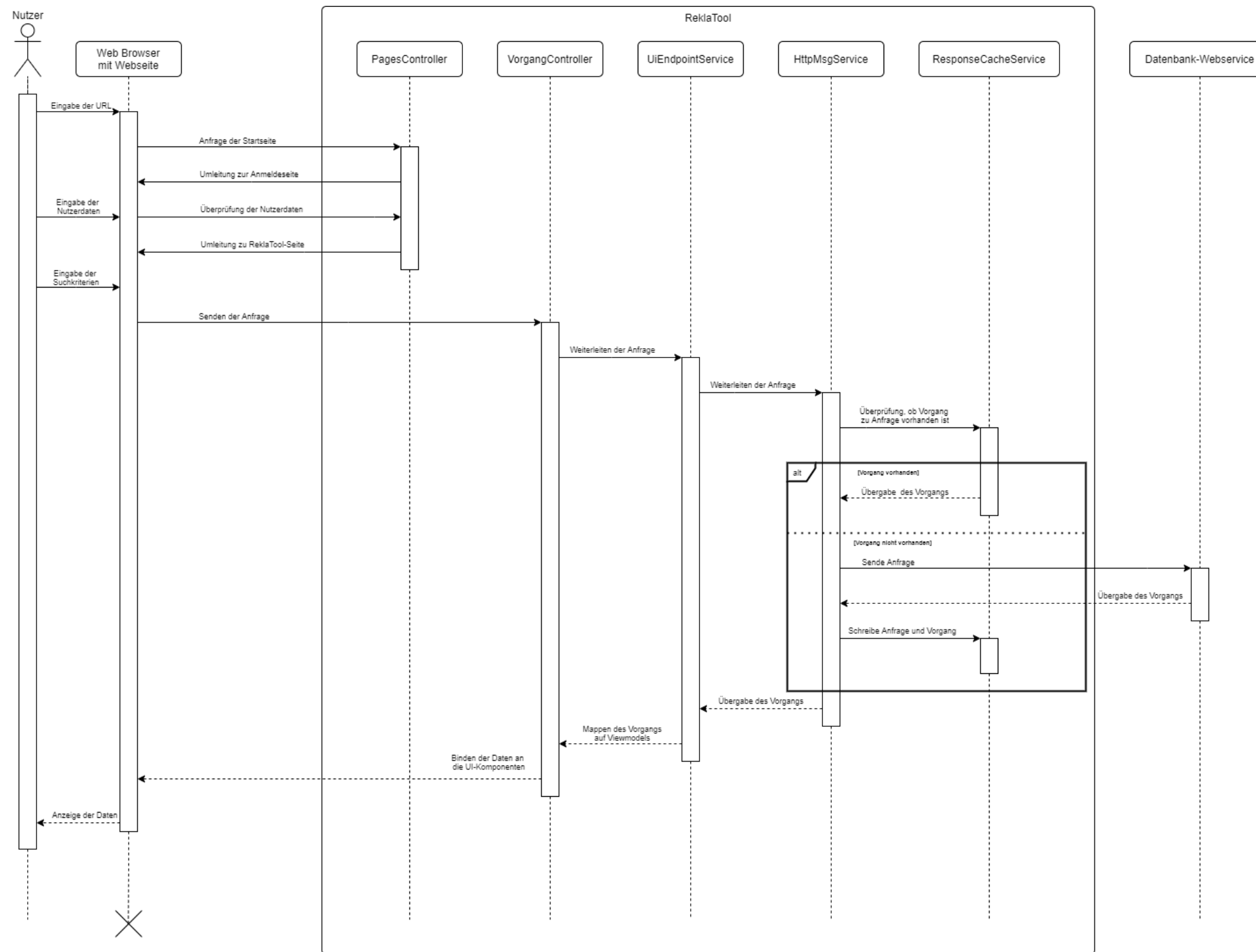
**A.9 Sequenzdiagramm**

Abbildung 6: Sequenzdiagramm

## A.10 Pflichtenheft (Auszug)

### Zielbestimmung

#### 1. Plattform

- 1.1. Die Anwendung wird in *C# 10.0* umgesetzt.
- 1.2. Das benutzte Framework ist *.NET 6.0*.
- 1.3. Als Webframework wird *ASP.Net Core MVC* genutzt.
- 1.4. Die Webanwendung läuft im Intranet der Icam.
- 1.5. Die Versionskontrolle erfolgt über *Git*.

#### 2. Benutzeroberfläche

- 2.1. Die Anwendung soll im aktuellsten Google Chrome Browser lauffähig sein.
- 2.2. Die Benutzeroberfläche wird mit Komponenten von *TelerikUI* umgesetzt.
- 2.3. Zur Erstellung werden die Programmiersprachen *C#* und *JS* verwendet.
- 2.4. Als Auszeichnungssprachen werden *HTML* und *CSS* benutzt.
- 2.5. Die Benutzeroberfläche wird für die Betrachtung auf einem Desktop-PC entworfen.
- 2.6. Daten werden in Tabellen strukturiert.
- 2.7. Teilbereiche sollen über Reiter erreichbar sein.
- 2.8. Die Suchfunktion soll einschränkbar sein.

#### 3. Geschäftslogik

- 3.1. Die Einbindung von *Services* erfolgt über *DI*.
- 3.2. Aus der API-Antwort heraus wird eine *XML*-Datei zum Download bereitgestellt.
- 3.3. Aus der API-Antwort heraus wird eine *PDF*-Datei zum Download bereitgestellt.
- 3.4. Suchergebnisse werden in einem *Cache* zwischengespeichert.

A.11 Klassendiagramm

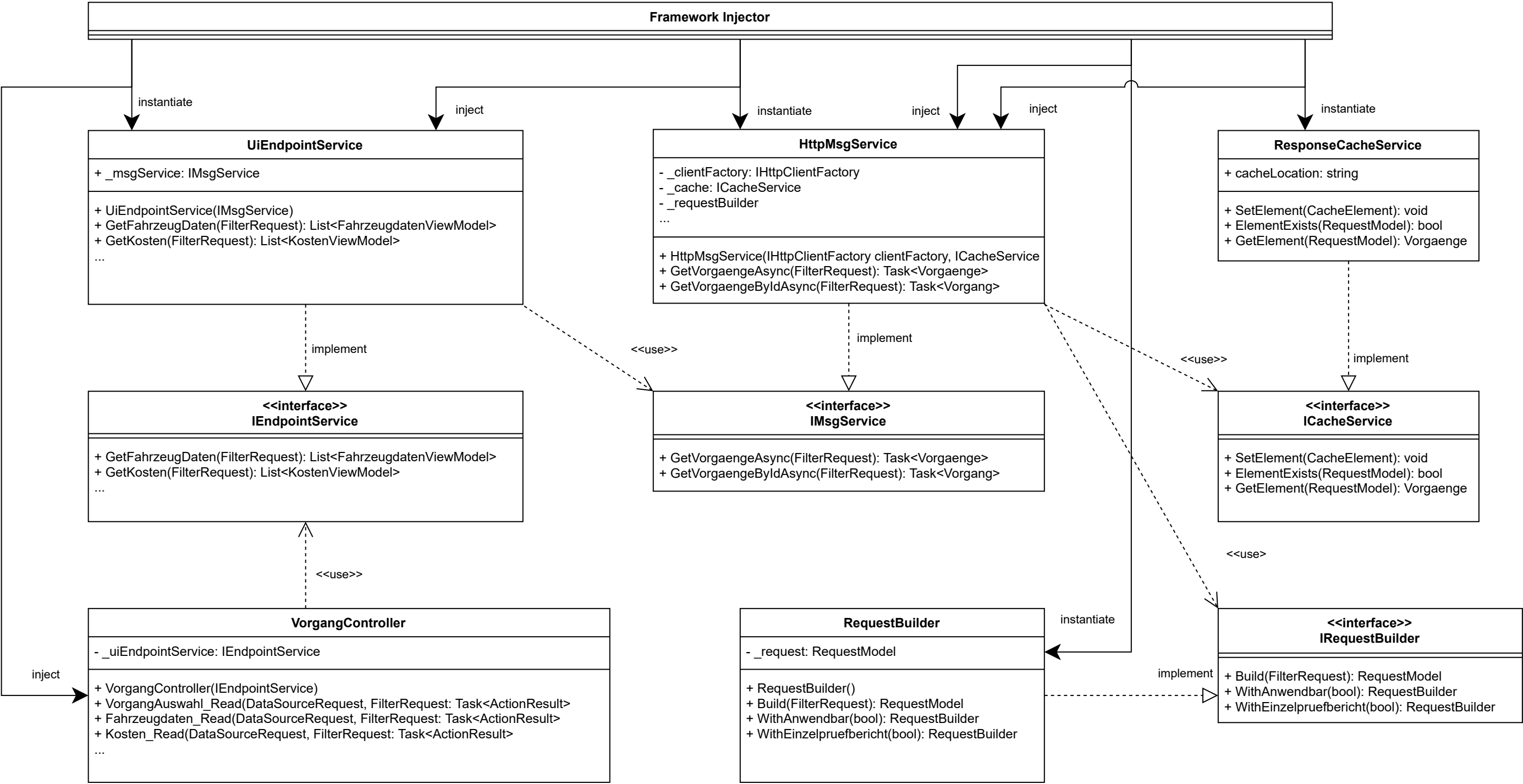


Abbildung 7: Klassendiagramm

## A.12 Datenmodell

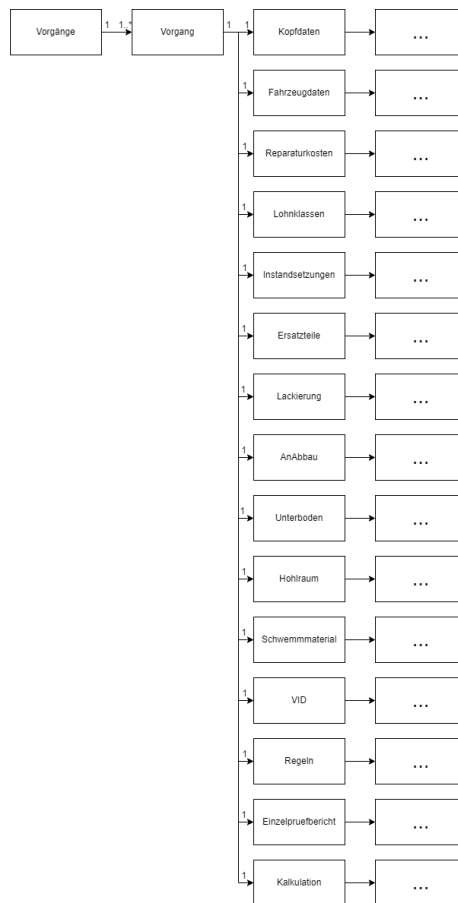


Abbildung 8: XML-Modell für Antwort

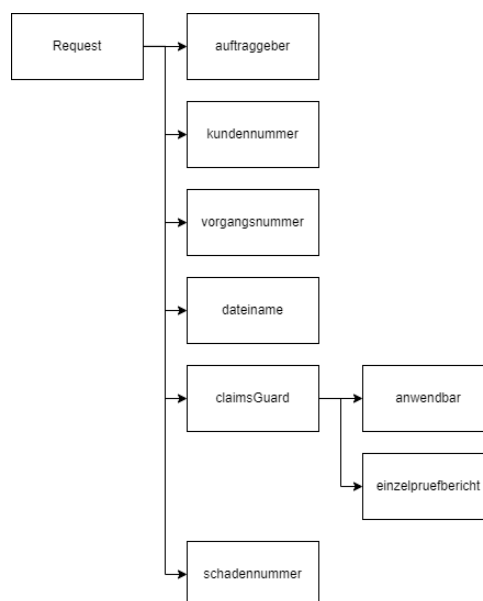


Abbildung 9: XML-Modell für Anfrage

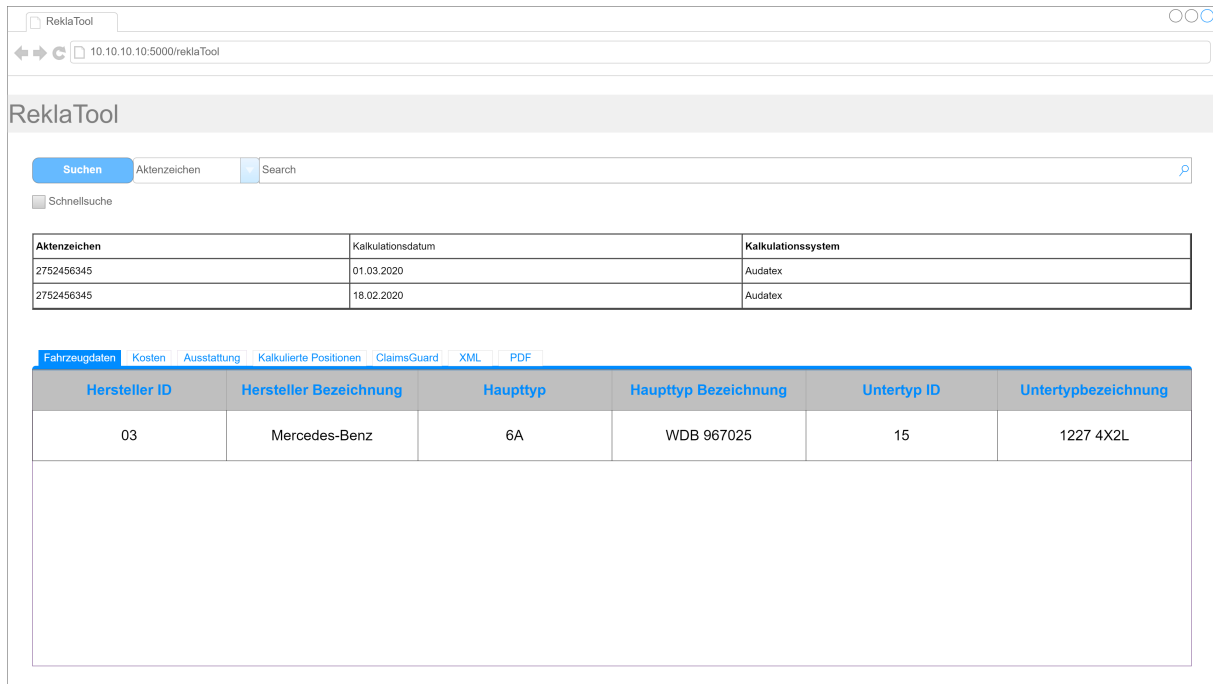
### A.13 DTO (Ausschnitt)

```
3 namespace ReklaTool.Models
4 {
5     public interface IKategorie
6     {
7         public List<Position> Position { get; set; }
8     }
9
10    [XmlRoot(ElementName = "Vorgaenge")]
11    public class Vorgaenge
12    {
13        [XmlElement(ElementName = "Vorgang")]
14        public List<Vorgang> Vorgang { get; set; }
15    }
16
17    [XmlRoot(ElementName = "Vorgang")]
18    public class Vorgang
19    {
20        [XmlElement(ElementName = "Kopfdaten")]
21        public Kopfdaten Kopfdaten { get; set; }
22
23        [XmlElement(ElementName = "Fahrzeugdaten")]
24        public Fahrzeugdaten Fahrzeugdaten { get; set; }
25
26        [XmlElement(ElementName = "Reparaturkosten")]
27        public Reparaturkosten Reparaturkosten { get; set; }
28
29        [XmlElement(ElementName = "Lohnklassen")]
30        public Lohnklassen Lohnklassen { get; set; }
31
32        [XmlElement(ElementName = "Instandsetzungen")]
33        public Instandsetzungen Instandsetzungen { get; set; }
```

Listing 1: ResponseModel.cs – XML-Annotationen



## A.14 Oberflächenentwurf



The mockup shows a web browser window with the address bar displaying '10.10.10.10:5000/reklaTool'. The application title is 'ReklaTool'. Below the title is a search bar with a 'Suchen' button and a 'Schnellsuche' checkbox. A table displays search results with columns: Aktenzeichen, Kalkulationsdatum, and Kalkulationssystem. Below this is a navigation bar with tabs: Fahrzeugdaten, Kosten, Ausstattung, Kalkulierte Positionen, ClaimsGuard, XML, and PDF. The main content area shows a table with columns: Hersteller ID, Hersteller Bezeichnung, Haupttyp, Haupttyp Bezeichnung, Untertyp ID, and Untertypbezeichnung. The table contains one row with data: 03, Mercedes-Benz, 6A, WDB 967025, 15, and 1227 4X2L.

Aktenzeichen	Kalkulationsdatum	Kalkulationssystem
2752456345	01.03.2020	Audatex
2752456345	18.02.2020	Audatex

Hersteller ID	Hersteller Bezeichnung	Haupttyp	Haupttyp Bezeichnung	Untertyp ID	Untertypbezeichnung
03	Mercedes-Benz	6A	WDB 967025	15	1227 4X2L

Abbildung 10: Mockup der Benutzeroberfläche

## A.15 View (Auszug)

```

19 <table width="100%">
20   <tr>
21     <td>
22       @(Html.Kendo().Button().Name("btn_submit").Content("Suchen").Events(e => e.Click("loadAuswahl"))
23         .Enable(false)
24     </td>
25     <td>
26       @(Html.Kendo().DropDownList().Name("dropDownAktenzeichen")
27         .BindTo(new List<string>()
28           {
29             "Vorgangsnummer",
30             "Auftragsnummer",
31             "Schadenummer",
32             "Dateiname"
33           })
34     </td>
35   <td width="100%">
36     <div class="suche">
37       @(Html.Kendo().TextBox()
38         .Name("textbox_Suchfeld")
39         .FillMode(FillMode.None)
40         .HtmlAttributes(new { @class = "suchfeld" })

```

```

41         .Placeholder("Suchbegriff: Auftragsnummer, Schadennummer, Filename oder
42             Vorgangsnummer")
43     )
44 </div>
45 </td>
46 </tr>
47 </table>
48 @Html.Kendo().CheckBox().Name("checkbox_Schnellsuche").Checked(true).Label("Schnellsuche")
49 <br/>
50 <div class="vorgangListe">
51     <hr/>
52     @(Html.Kendo().Grid<VorgangInfoModel>()
53         .Name("vorgangAuswahl")
54         .AutoBind(false)
55         .NoRecords(true)
56         .Sortable()
57         .Columns(columns =>
58             {
59                 columns.Bound(e => e.Aktenzeichen);
60                 columns.Bound(e => e.Kalkulationsdatum);
61                 columns.Bound(e => e.Kalkulationssystem);
62             })
63         .DataSource(dataSource => dataSource
64             .WebApi()
65             .ServerOperation(false)
66             .PageSize(5)
67             .Read(read => read
68                 .Data("getSearchParametersFast")
69                 .Action("VorgangAuswahl_Read", "Vorgang")
70             )
71             .Sort(c=>c.Add("Kalkulationsdatum").Descending())
72             .Events(ev=>ev.Error("vorgangNotFound"))
73         )
74         .Pageable()
75         .Selectable(sel => sel.Mode(GridSelectionMode.Single))
76         .Events(ev => ev.Change("VorgangSelected").DataBound("VorgangAuswahlGeladen"))
77     )
78 </div>

```

Listing 2: View-Komponenten – Checkbox und Auswahlliste

## A.16 Screenshots der Anwendung

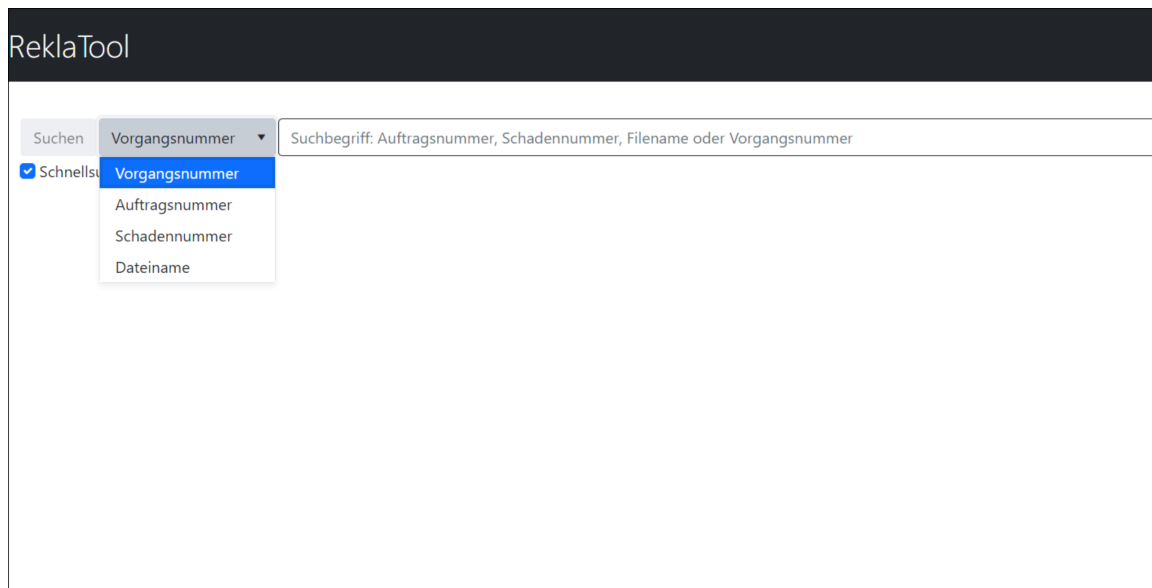


Abbildung 11: Eingabe des Aktenzeichens und Auswahl des Typs (Ausschnitt)

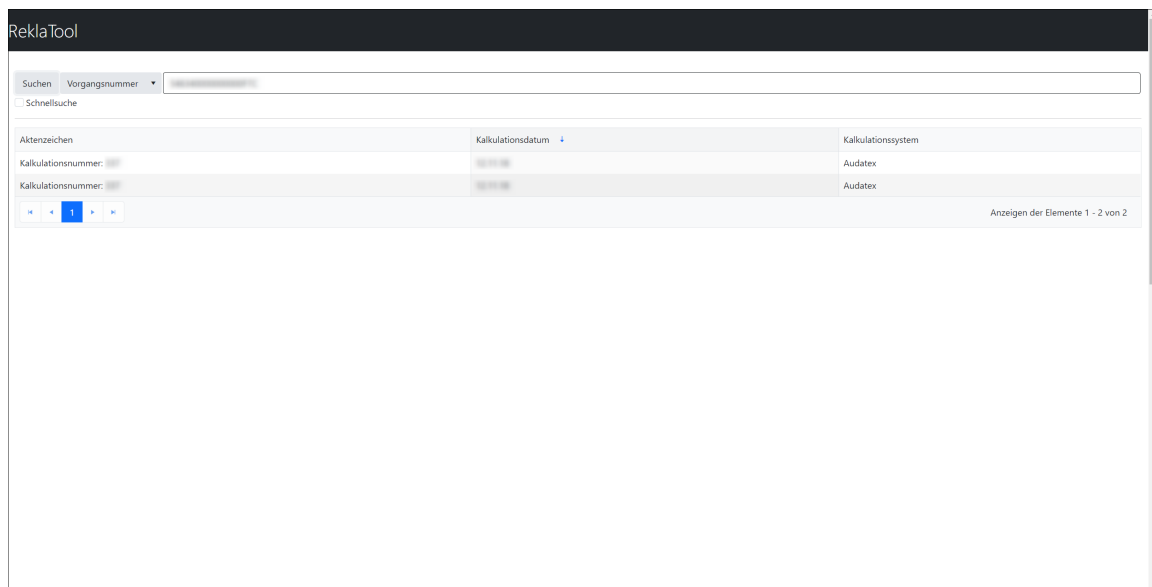


Abbildung 12: Liste der Vorgänge zum Aktenzeichen

ReklaTool

Suchen Vorgangsnummer

☐ Schnellsuche

Aktenzeichen	Kalkulationsdatum	Kalkulationssystem
Kalkulationsnummer: <input type="text"/>	10.11.16	Audatex
Kalkulationsnummer: <input type="text"/>	10.11.16	Audatex

Anzeigen der Elemente 1 - 2 von 2

Aktueller Vorgang

Fahrzeugdaten Kosten Ausstattung Kalkulierte Positionen ClaimsGuard PDF Download

Kategorie	Leitnummer	Bauart	Bezeichnung	Betrag	Materialeigenschaft	Platzhalter	Repart	Teilenummer	Stern
<b>Kategorie: AnAbbau</b>									
AnAbbau	1496		SCHARNIER TÜR U R	52.00			N		
<b>Kategorie: Ersatzteile</b>									
Ersatzteile	0292	M	STOSSFAENGER V R	57.11	2		E	967 885 0703 7G99	
Ersatzteile	0487	M	EMBLEM "ATEGO"	18.44	0		E	967 817 0120	
Ersatzteile	0672	M	KONSOLE V R	13.91	2		E	973 826 0131 7C72	
Ersatzteile	0678	M	BLINKLEUCHTE SEITL R	15.85	0		E	967 820 1221	
Ersatzteile	0888	M	LUFTFUEHRUNG ABD R	15.44	2		E	967 884 0374	
Ersatzteile	0892	M	HALTERUNG ABDECK U R	3.16	0		E	967 884 0715	
Ersatzteile	0894	M	BLLENDE WINDLEIT A R	69.42	2		E	967 884 1022	

Abbildung 13: Darstellung der Daten in einer Tabelle

ReklaTool

Suchen Vorgangsnummer

☐ Schnellsuche

Aktenzeichen	Kalkulationsdatum	Kalkulationssystem
Kalkulationsnummer: <input type="text"/>	10.11.16	Audatex
Kalkulationsnummer: <input type="text"/>	10.11.16	Audatex

Anzeigen der Elemente 1 - 2 von 2

Aktueller Vorgang

Fahrzeugdaten Kosten Ausstattung Kalkulierte Positionen ClaimsGuard PDF Download

1 of 1 Automatic W...

**PRÜFBERICHT\***

Kalkulations-Nr.  Schadennummer

**claims guard**

**Diese Werkstatt ist nicht als Partnerwerkstatt angelegt.**

\* Bei allen Preiskategorien handelt es sich um Festbeträge

ÜBERSICHT	ARBEITSLOHN	NEBENKOSTEN	LACKIERUNG	ERSATZTEILE
REPARATURKOSTEN				
SVS	UK 1	UK 2	UK 3	LACK
Kalkuliert				
BEREICH LACKIERUNG	LACKSYSTEM	LACKMATERIAL	LACKAUSFÜHRUNG	
Kalkuliert	Hersteller	prozentual 40 %	Zusatzschicht	

Abbildung 14: Integrierte PDF-Anzeige

## A.17 Übergabeprotokoll

IcamSystems GmbH



### Abnahmeprotokoll

Gegenstand	Software-Abnahme		
Projekt	ReklaTool		
Kunde	IcamSystems GmbH		
Prüfer	Danny Sotzny		
Daten	ReklaTool.sln und Projektordner		
Zielplattform	Microsoft Windows, Google Chrome Browser		
Komponente	Webanwendung		
Entwickler	Ingolf Schieck		
Revision	1.0		
Code-Review	erfolgreich	Integrationstest	erfolgreich

### Abnahmeergebnis

Das Abnahmeobjekt wurde ohne Mängelanzeige geprüft. Die Abnahme erfolgt vorbehaltlos.

Leipzig, 26.10.2022  
\_\_\_\_\_  
Ort, Datum

Leipzig, 26.10.2022  
\_\_\_\_\_  
Ort, Datum

\_\_\_\_\_  
Unterschrift Prüfer

\_\_\_\_\_  
Unterschrift Entwickler

## A.18 Entwicklerdokumentation (Auszug)

### Einbindung von Services

In der `Program.cs` werden die Services beim *Servicecontainer* des Frameworks registriert (siehe Codezeilen 7-18). Dazu wird der Servicebuilder mit `builder.Services` aufgerufen. Dessen Methode `.AddScoped<T>` fügt dem Container den Service direkt, oder über ein Interface hinzu.

```
5 var builder = WebApplication.CreateBuilder(args);
6 // Add services to the container.
7 builder.Services.AddControllersWithViews()
8     // Maintain property names during serialization . See:
9     // https://github.com/aspnet/Announcements/issues/194
10    .AddNewtonsoftJson(options => options.SerializerSettings.ContractResolver
11        = new Newtonsoft.Json.Serialization.DefaultContractResolver());
12 // Add Kendo UI services to the services container"
13 builder.Services.AddKendo();
14 builder.Services.AddHttpClient();
15 builder.Services.AddScoped<IMsgService, HttpMsgService>();
16 builder.Services.AddScoped<IEndpointService, UiEndpointService>();
17 builder.Services.AddScoped<IRequestBuilder, RequestBuilder>();
18 builder.Services.AddScoped<ICacheService, ResponseCacheService>();
19 builder.Services.AddKeycloakAuthentication(builder.Configuration);
20
21 var app = builder.Build();
```

Program.cs – Services

Einbindung der Middleware.

```
31 app.UseHttpsRedirection();
32 app.UseStaticFiles();
33 app.UseRouting();
34 app.UseKeycloakAuthentication();
35 app.UseFrontendToken(); // Induziert am Ende den Token auf die Seite
36 app.UseAuthorization();
```

Program.cs – Middleware

## Aufbau des HttpMsgService

Die Datei HttpMsgService.cs beinhaltet das Interface IMsgService.

```

7  public interface IMsgService
8  {
9      Task<Vorgaenge?> GetVorgaengeAsync(FilterRequest model);
10     Task<Vorgang> GetVorgangByldAsync(FilterRequest model);
11 }

```

HttpMsgService.cs – Interface

Die Injektion der Services über den Konstruktor. Diese werden automatisch vom Injektor des Framework bereitgestellt.

```

18  public HttpMsgService(IConfiguration config, IHttpClientFactory clientFactory, ICacheService cache,
19                          IRequestBuilder requestBuilder)
20  {
21      _configuration = config;
22      _clientFactory = clientFactory;
23      _cache = cache;
24      _requestBuilder = requestBuilder;
25  }

```

HttpMsgService.cs – Konstruktorinjektion

Services werden durch private Felder gehalten.

```

14  private readonly IConfiguration _configuration;
15  private readonly IHttpClientFactory _clientFactory;
16  private readonly ICacheService _cache;
17  private readonly IRequestBuilder _requestBuilder;

```

HttpMsgService.cs – Services

Die Anfragen an die Datenbank-API werden über die Methode GetVorgaengeAsync gestellt. Das RequestModel für die Anfrage wird über den \_requestBuilder erstellt. Dieser kann über ein *Fluent-Interface* konfiguriert werden.

```

25  public async Task<Vorgaenge?> GetVorgaengeAsync(FilterRequest model)
26  {
27      RequestModel anfrage = _requestBuilder
28          .WithAnwendbar(!model.IsSchnellsuche)
29          .WithEinzelpruefbericht(!model.IsSchnellsuche)
30          .Build(model);

```

GetVorgaengeAsync – Anfrage-Model

Das Anfrage-Object wird zunächst an den `_cache` geleitet. Findet dieser ein entsprechendes Element, so gibt die Methode dieses zurück.

```
31         if (_cache.ElementExists(anfrage))
32         {
33             return _cache.GetElement(anfrage);
34         }
```

GetVorgaengeAsync – Cache-Abfrage

Falls kein Element im Cache vorhanden ist, wird zunächst eine Instanz eines HTTP-Clients erstellt.

```
35         else
36         {
37             HttpClient client = _clientFactory.CreateClient();
```

GetVorgaengeAsync – HTTP-Client

Danach wird das Anfrage-Objekt serialisiert und zusammen mit der API-Adresse und dem Typ der Anfrage in ein `HttpRequestMessage`-Objekt verpackt. Dieses wird dem Client übergeben und an die API versendet.

```
39         var x4Path = _configuration.GetValue<string>("ReklaConfig:X4Webaddress");
40         var requestMsg = new HttpRequestMessage(HttpMethod.Post, x4Path)
41         {
42             Content = new StringContent(anfrage.ToXML())
43         };
44
45         HttpResponseMessage response = await client.SendAsync(requestMsg);
```

GetVorgaengeAsync – API-Anfrage

Der Inhalt der empfangenen Antwort wird deserialisiert. Danach wird das Antwort-Objekt zusammen mit Anfrage-Objekt direkt in den Cache geschrieben. Abschließend gibt die Methode `GetVorgaengeAsync` ein Objekt mit den angefragten Vorgängen zurück.

```
47         var content = await response.Content.ReadAsStringAsync();
48
49         var serializer = new XmlSerializer(typeof(Vorgaenge));
50         using TextReader reader = new StringReader(content);
51         Vorgaenge? vorgaenge = (Vorgaenge)serializer.Deserialize(reader);
52
53         _cache.SetElement(new CacheElement { Anfrage = anfrage, Vorgang = vorgaenge });
54
55         return vorgaenge;
56     }
57 }
```

GetVorgaengeAsync – Antwort der API



## A.19 Benutzerdokumentation (Ausschnitt)

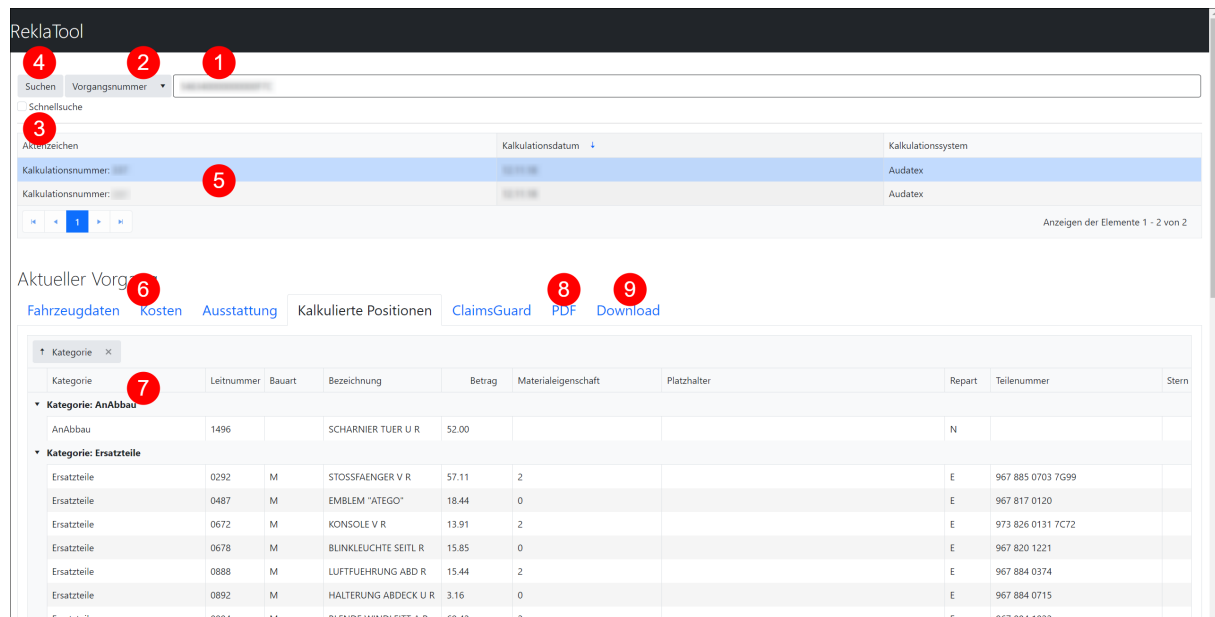


Abbildung 15: Ansicht des ReklaTool

### Benutzung des ReklaTool

1. Aktenzeichen eingeben.
2. Typ des Aktenzeichens auswählen.
3. Option zur Auswahl der Schnellsuche. Wenn ausgewählt, werden ClaimsGuard-Regeln und PDF nicht mitgeschickt.
4. Button zum Starten der Suchanfrage.
5. Vorgang auswählen.
6. Auswahl einer Kategorie durch klicken auf einen der Reiter.
7. In der Tabellenansicht können Elemente sortiert und gruppiert werden. Zum Gruppieren den Titel einer Spalte in die Zeile darüber ziehen. Zum Sortieren auf den Spaltentitel mit dem Sortierkriterium klicken.
8. Anzeigen des integrierten PDF-Readers. Möglichkeit zum Download der PDF.
9. Download der Strukturierten Daten zum Vorgang im XML-Format.