

# Prüfungsteil A

Prüfling (private Anschrift):

Leon Brachwitz  
Mecklenburger Straße 6  
49456 Bakum

Ausbildungsbetrieb:

ALTE OLDENBURGER Krankenversicherung AG  
Alte-Oldenburger-Platz 1  
49377 Vechta

## Bestätigung über durchgeführte Projektarbeit

diese Bestätigung ist mit der Projektdokumentation einzureichen

Ausbildungsberuf (bitte unbedingt angeben):

Fachinformatiker für Anwendungsentwicklung

Projektbezeichnung:

AO-Pflegerechner


Projektbeginn: 02.03.2020 Projektfertigstellung: 13.04.2020 Zeitaufwand in Std.: 70

## Bestätigung der Ausbildungsfirma:


Wir bestätigen, dass der/die Auszubildende das oben bezeichnete Projekt einschließlich der Dokumentation im Zeitraum

vom: 02.03.2020 bis: 13.04.2020 selbständig ausgeführt hat.

Projektverantwortliche(r) in der Firma:

Stefan	Macke	04441 905 173	
Vorname	Name	Telefon	Unterschrift

Ausbildungsverantwortliche(r) in der Firma:

Stefan	Macke	04441 905 173	
Vorname	Name	Telefon	Unterschrift

## Eidesstattliche Erklärung:

Ich versichere, dass ich das Projekt und die dazugehörige Dokumentation selbständig erstellt habe.

Ort und Datum: Vechta, 04.05.2020 Unterschrift des Prüflings: Brachwitz



Abschlussprüfung Sommer 2020

Fachinformatiker für Anwendungsentwicklung  
Dokumentation zur betrieblichen Projektarbeit

## AO-Pflegerechner

Web-Anwendung für die Beratung in der Pflegezusatzversicherung

Abgabetermin: Vechta, den 06.05.2020

**Prüfungsbewerber:**

Leon Brachwitz  
Mecklenburger Straße 6  
49456 Bakum



**Ausbildungsbetrieb:**

ALTE OLDENBURGER Krankenversicherung AG  
Alte-Oldenburger-Platz 1  
49377 Vechta

## 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 Projektbeschreibung . . . . .	1
1.3 Projektziel . . . . .	1
1.4 Projektbegründung . . . . .	2
1.5 Projektschnittstellen . . . . .	2
1.6 Abweichungen vom Projektantrag . . . . .	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 . . . . .	6
3.2.3 Amortisationsdauer . . . . .	6
3.3 Nicht-monetäre Aspekte . . . . .	7
3.4 Anwendungsfälle . . . . .	7
3.5 Lastenheft . . . . .	8
<b>4 Entwurfsphase</b>	<b>8</b>
4.1 Entwurf der Benutzeroberfläche . . . . .	8
4.2 Zielplattform . . . . .	8
4.3 Architekturdesign . . . . .	9
4.4 Datenmodell . . . . .	10
4.5 Geschäftslogik . . . . .	10
4.6 Maßnahmen zur Qualitätssicherung . . . . .	11
4.7 Deployment . . . . .	11
4.8 Pflichtenheft . . . . .	12

<b>5 Implementierungsphase</b>	<b>12</b>
5.1 Implementierung der Benutzeroberfläche . . . . .	12
5.2 Implementierung der Datenstrukturen . . . . .	13
5.3 Implementierung der Geschäftslogik . . . . .	13
5.4 Testen der Anwendung . . . . .	14
<b>6 Abnahme- und Einführungsphase</b>	<b>14</b>
<b>7 Dokumentation</b>	<b>14</b>
<b>8 Fazit</b>	<b>15</b>
8.1 Soll-/Ist-Vergleich . . . . .	15
8.2 Lessons Learned . . . . .	15
8.3 Ausblick . . . . .	15
<b>Literaturverzeichnis</b>	<b>16</b>
<b>A Anhang</b>	<b>i</b>
A.1 Detaillierte Zeitplanung . . . . .	i
A.2 Ressourcenplanung . . . . .	ii
A.3 Screenshot der Exceltabelle . . . . .	iii
A.4 Amortisation . . . . .	iv
A.5 Use-Case-Diagramm . . . . .	v
A.6 Lastenheft . . . . .	vi
A.7 Oberflächenentwürfe . . . . .	vii
A.8 Komponentendiagramm . . . . .	ix
A.9 Entity-Relationship-Model . . . . .	x
A.10 Tabellenmodell . . . . .	xi
A.11 Aktivitätsdiagramm . . . . .	xii
A.12 Testabdeckung Ansicht aus SonarQube . . . . .	xiii
A.13 Verteilungsdiagramm . . . . .	xiv
A.14 Jenkinsfile (Auszug) . . . . .	xv
A.15 JBoss Konfiguration (Auszug) . . . . .	xvi
A.16 Pflichtenheft . . . . .	xvii
A.17 Listing von JSF-Code . . . . .	xviii
A.18 Listing von JavaScript-Code . . . . .	xix
A.19 Screenshots der fertigen Anwendung . . . . .	xx
A.20 Listing von Java-Code . . . . .	xxi
A.21 Benutzerdokumentation (Auszug) . . . . .	xxiii
A.22 Entwicklerdokumentation (Auszug) . . . . .	xxiv
A.23 Klassendiagramm der Domäne (Auszug) . . . . .	xxv

## Abbildungsverzeichnis

1	Screenshot der Exceltabelle . . . . .	iii
2	Amortisation . . . . .	iv
3	Use-Case-Diagramm . . . . .	v
4	Anzeige der Benutzeroberfläche im Browser . . . . .	vii
5	Ansicht der generierten PDF-Datei . . . . .	viii
6	Komponentendiagramm . . . . .	ix
7	Entity-Relationship-Model . . . . .	x
8	Tabellenmodell . . . . .	xi
9	Aktivitätsdiagramm des Use-Cases Zahlbeitrag ermitteln . . . . .	xii
10	Testabdeckung Ansicht aus SonarQube . . . . .	xiii
11	Verteilungsdiagramm . . . . .	xiv
12	Ansicht der Login-Seite . . . . .	xx
13	Ansicht Anwendung . . . . .	xx
14	Auszug aus der Benutzerdokumentation . . . . .	xxiii
15	Auszug aus der Entwicklerdokumentation . . . . .	xxiv
16	Auszug des Klassendiagramms der Domäne . . . . .	xxv

## Tabellenverzeichnis

1	Zeitplanung (kurz) . . . . .	3
2	Kostenaufstellung . . . . .	6
3	Zeiteinsparung . . . . .	7

## Listings

1	Auszug aus dem Jenkinsfile . . . . .	xv
2	JBoss-Konfigurationsdatei . . . . .	xvi
3	Datei: <code>index.xhtml</code> . . . . .	xviii
4	Datei: <code>diagramm.js</code> . . . . .	xix
5	Klasse: <code>PflegelueckeServiceSollte</code> . . . . .	xxi
6	Klasse: <code>PflegelueckeService</code> . . . . .	xxii

## Abkürzungsverzeichnis

<b>AJAX</b>	Asynchronous JavaScript and XML
<b>AO</b>	ALTE OLDENBURGER Krankenversicherung AG
<b>API</b>	Application Programming Interface
<b>BaFin</b>	Bundesanstalt für Finanzdienstleistungsaufsicht
<b>BAP</b>	Beitragsanpassung
<b>CD</b>	Continuous Deployment
<b>CDI</b>	Contexts and Dependency Injection
<b>CSS</b>	Cascading Style Sheets
<b>DTO</b>	Datentransferobjekt
<b>EAP</b>	Enterprise Application Platform
<b>ERM</b>	Entity-Relationship-Model
<b>HTML</b>	Hypertext Markup Language
<b>IDV</b>	Individuelle Datenverarbeitung
<b>IDV-Leitlinie</b>	Leitlinie für die individuelle Datenverarbeitung
<b>Jakarta EE</b>	Jakarta Enterprise Edition
<b>JPA</b>	Java Persistence Application Programming Interface ( <a href="#">API</a> )
<b>JSF</b>	Java Server Faces
<b>JSON</b>	JavaScript Object Notation
<b>MVC</b>	Model View Controller
<b>Orga</b>	Fachabteilung Organisation & Vertrieb
<b>PDF</b>	Portable Document Format
<b>POJO</b>	Plain Old Java Object
<b>PR</b>	Pull-Request
<b>PT</b>	Personentag
<b>REST</b>	Representational State Transfer
<b>SCM</b>	Sourcecode Management
<b>TDD</b>	Test Driven Development
<b>VAIT</b>	Versicherungsaufsichtliche Anforderungen an die IT
<b>XHTML</b>	Extensible Hypertext Markup Language
<b>YAML</b>	YAML Ain't Markup Language

## 1 Einleitung

In der folgenden Projektdokumentation wird der Ablauf des IHK-Abschlussprojektes, das der Autor im Rahmen seiner Ausbildung zum Fachinformatiker für Anwendungsentwicklung durchgeführt hat, erläutert.

### 1.1 Projektumfeld

Ausbildungsbetrieb ist die ALTE OLDENBURGER Krankenversicherung AG (AO). Die AO ist eine private Krankenversicherung mit Sitz in Vechta. Zur Zeit beschäftigt die AO 256 Mitarbeiter.<sup>1</sup> Der Leistungsumfang umfasst private Krankheitskostenvollversicherungen, Zusatzversicherungen zur gesetzlichen Krankenversicherung sowie Pflegezusatzversicherungen. Auftraggeber des Projektes ist die Fachabteilung Organisation & Vertrieb (Orga) der AO. Zu den Aufgaben der Orga gehören die Beratung von Interessenten und Vermittlern sowie die Angebotserstellung.

### 1.2 Projektbeschreibung

Im Bereich der Pflegezusatzversicherung vertreibt die AO einen Tarif unter dem Namen *PflegeSchutz*. Als pflegebedürftig gelten Personen, die gesundheitlich bedingte Beeinträchtigungen der Selbstständigkeit oder der Fähigkeiten aufweisen und deshalb der Hilfe durch andere bedürfen.<sup>2</sup> Hierzu gehören beispielsweise die Mobilität, kognitive und kommunikative Fähigkeiten oder die Fähigkeit der Selbstversorgung. Im Falle der Pflegebedürftigkeit und dem damit verbundenen Antrag auf Leistung, greift die gesetzliche Pflegeversicherung. Sie leistet abhängig von der Art (ambulant oder stationär) einen festgelegten Beitrag. Allein durch die gesetzliche Absicherung können die aufzubringenden Pflegekosten nicht gedeckt werden. Um einer finanziellen Belastung im Pflegefall vorzubeugen, kann bei der AO der Tarif *PflegeSchutz* sowie der *PflegeBahr* abgeschlossen werden. Der *PflegeBahr* ist eine staatlich geförderte ergänzende Pflegezusatzversicherung mit dem Ziel, eine Basisabsicherung zu schaffen. Der Tarif *PflegeSchutz* der AO soll neben der gesetzlichen Pflegeversicherung und der Basisabsicherung durch den *PflegeBahr* die restlichen Pflegekosten abdecken bzw. die Pflegelücke reduzieren. Der Vertrieb dieser Tarife erfolgt bei der AO im Makler- und Direktgeschäft. Im Direktgeschäft übernimmt dies die Orga.

Das Projekt beschäftigt sich mit der Bereitstellung einer Software, die die Fachabteilung und die Makler bei der Berechnung der Zahlbeiträge und der Beratung im Direkt- und Fernabsatzgeschäft der Pflegezusatzversicherung unterstützt.

### 1.3 Projektziel

Das Ziel des Projekts ist, eine Web-Anwendung mit grafischer Benutzeroberfläche und Datenbank-anbindung zu entwickeln. Die Anwendung soll den Fachbereich bei der Kalkulation des Beitrags in der Pflegezusatzversicherung und der Beratung von Kunden unterstützen und im Internet aufrufbar sein. Das jährliche Aktualisieren der Daten soll in der IT-Abteilung erfolgen. Die Daten, die

---

<sup>1</sup>Vgl. ALTE OLDENBURGER [2019].

<sup>2</sup>Vgl. BUNDESMINISTERIUM DER JUSTIZ UND FÜR VERBRAUCHERSCHUTZ [2019].

## 1 Einleitung

---

für die Kalkulation notwendig sind, sollen in einer Datenbank abgelegt werden. Durch Continuous Integration soll der Build automatisiert werden, um die Wartbarkeit und das Ausrollen von neuen Versionen zu vereinfachen.

### 1.4 Projektbegründung

Der Vertrieb von Pflegezusatzversicherungen gehört zum Tagesgeschäft der AO. Aktuell wird zur Unterstützung bei der Beratung und beim Abschluss der Versicherungen eine vom Fachbereich erstellte Excel-Datei verwendet. Eine im Konzern geltende Leitlinie für die individuelle Datenverarbeitung ([IDV-Leitlinie](#)) definiert, dass die Anzahl von dezentralen IT-Lösungen, die nicht zur Anwendungslandschaft der AO gehören, reduziert werden soll. Grund hierfür ist beispielsweise die hohe Fehleranfälligkeit sowohl bei der Entwicklung als auch bei der regelmäßigen Nutzung durch verschiedene Mitarbeiter, z. B. durch versehentliche Änderung von Formeln und Daten. Des Weiteren ist ein Nachweis der Funktionsfähigkeit durch Tests und eine nachvollziehbare Dokumentation bei Änderungen an der IDV-Anwendung erforderlich. Diese Anforderungen werden von der Bundesanstalt für Finanzdienstleistungsaufsicht ([BaFin](#)) im Rundschreiben Versicherungsaufsichtliche Anforderungen an die IT ([VAIT](#)) definiert und vorgeschrieben. Die Umsetzung für IDV-Anwendungen ist mit erheblichem Aufwand verbunden und soll beispielsweise durch automatisierte Tests reduziert werden. Außerdem soll der Aufwand für die Wartung und Aktualisierung der Berechnungsgrundlagen nicht im Fachbereich, sondern in der IT-Abteilung des Unternehmens liegen. Ein weiterer Grund für die Ablösung der Excel-Datei ist, dass eine optisch ansprechende Web-Anwendung bei der Demonstration vor dem Kunden einen guten Eindruck hinterlässt.

### 1.5 Projektschnittstellen

Die Berechnungsgrundlagen für die Tarife werden von der Mathematik-Abteilung kalkuliert und in einer sich im internen Firmennetz befindlichen Oracle-Datenbank abgelegt. Der Pflegerechner wird sich außerhalb des Firmennetzes befinden. Da von außerhalb nicht auf die Server im internen Netz der AO zugegriffen werden kann, muss ein Export der Daten aus der internen Oracle-Datenbank heraus in eine MariaDB erfolgen.

Die Benutzer der Web-Anwendung sollen eng in den Entwicklungsprozess eingebunden werden. Durch ständigen Austausch soll eine schnelle Reaktion auf Wünsche und Änderungen erfolgen können. Insbesondere bei der Gestaltung der Oberfläche sollen die Benutzer mit einbezogen werden. Hierdurch kann Zeit in der Einführungsphase in Form von Benutzerschulungen eingespart werden.

### 1.6 Abweichungen vom Projektantrag

Im Projektantrag wurde in der Abnahmephase ein zweistündiges Code-Review eingeplant. Da sich der interne Entwicklungsprozess in der Zeit zwischen Antrag und Beginn des Projektes geändert hat, entfällt dieses. Stattdessen wird mit Pull-Request ([PR](#)) gearbeitet. Der genaue Ablauf des Prozesses wird in Abschnitt [2.3 \(Entwicklungsprozess\)](#) behandelt.



## 2 Projektplanung

In diesem Kapitel geht es darum, wie das weitere Vorgehen bestimmt wurde. Das Projekt wurde dazu zeitlich geplant, alle benötigten Ressourcen ermittelt und der Entwicklungsprozess bestimmt.

### 2.1 Projektphasen

Für die Umsetzung des Projekts standen 70 Stunden zur Verfügung. Im Rahmen der Projektplanung wurden die Stunden auf verschiedene Phasen aufgeteilt und somit der gesamte Projektablauf abgebildet. Eine grobe Zeitplanung mit den Hauptphasen kann [Tabelle 1](#) entnommen werden. Eine detaillierte Zeitplanung mit den einzelnen Schritten der jeweiligen Phase befindet sich im [Anhang A.1: Detaillierte Zeitplanung](#) auf Seite [i](#).

Projektphase	Geplante Zeit
Analysephase	7 h
Entwurfsphase	11 h
Implementierungsphase inkl. Tests	42 h
Abnahme und Einführung	5 h
Dokumentation	5 h
<b>Gesamt</b>	<b>70 h</b>

Tabelle 1: Zeitplanung (kurz)

### 2.2 Ressourcenplanung

Im Anschluss an die Zeitplanung wurden alle benötigten Ressourcen im [Anhang A.2: Ressourcenplanung](#) auf Seite [ii](#) aufgelistet. Hier wurde neben den Hard- und Softwareressourcen auch das Personal mit aufgenommen. Bei der Auswahl der Software wurde neben der Architekturrichtlinie der [AO](#) darauf geachtet, dass die Software entweder kostenfrei (Open-Source) zur Verfügung steht oder bereits Lizenzen im Unternehmen vorhanden sind. Dadurch sollten die anfallenden Projektkosten so gering wie möglich gehalten werden. Beispielsweise wurde für das Testen der Oberfläche das Open-Source Framework Arquillian ausgewählt und als Application-Server der *JBoss*, da hierfür bereits Lizenzen vorhanden waren.

### 2.3 Entwicklungsprozess

Vor der Implementierung des Projektes musste ein geeigneter Entwicklungsprozess ausgewählt werden, um eine definierte Vorgehensweise bei der Entwicklung festzulegen. Da die Anforderungen mit der [Orga](#) eindeutig definiert und festgelegt wurden, sollte sich die Umsetzung des Projekts an den Phasen des Wasserfall-Modells orientieren. In der Implementierungsphase sollte eine inkrementelle Entwicklung verfolgt werden. Hierzu wurde mit Hilfe der Versionsverwaltung *Git* und der Projektmanagementsoftware *Redmine* folgender Prozess eingehalten. Die Anwendungsfälle aus der [Orga](#), die noch in [Abschnitt 3.4 \(Anwendungsfälle\)](#) behandelt werden, werden als einzelne Features definiert. Zu jedem dieser Features wurde in Redmine ein Ticket angelegt. In Git gab es mehrere

### 3 Analysephase

---

Branches. Der Stand, der sich auf der produktiven Umgebung befand, befand sich auf dem Branch **master**. Noch nicht abgenommener bzw. durch die Fachabteilung getesteter Code befand sich auf dem Branch **qs**. Für die einzelnen Features wurden eigene Branches erstellt. Wurde während der Implementierung ein Feature bzw. Inkrement fertiggestellt, wurde in Git ein **PR** erstellt. Ein Mitarbeiter aus der IT-Abteilung musste ein Review des Codes durchführen. Wurde der **PR** beim Review abgenommen, konnte das Inkrement auf **qs** übertragen werden. Nun konnte ein Sachbearbeiter aus der Fachabteilung testen. War dieser zufrieden, stellte er das Ticket in den Status *abgenommen* und das Feature wurde auf **master** übertragen. Anschließend konnte das nächste Inkrement implementiert werden.

In der Implementierungsphase sollte die Praktik des Test Driven Development (**TDD**) angewendet werden. Hierbei wurden die JUnit-Tests vor dem eigentlichen Quellcode geschrieben. Die Vorteile, die sich hieraus ergaben, werden im Abschnitt 4.6 (**Maßnahmen zur Qualitätssicherung**) erläutert.

Durch den ständigen Austausch mit der Fachabteilung und die Tests der einzelnen Features konnte die Zeit für die Abnahme der fertigen Anwendung deutlich reduziert werden. Des Weiteren wurden Fehler frühzeitig entdeckt und konnten so noch während der Implementierungsphase behoben werden.

## 3 Analysephase

Um zu ermitteln, was sich die Fachabteilung wünscht und wie der derzeitige Stand ist, wurde eine Analyse durchgeführt. Neben der Ermittlung der Anwendungsfälle wurde außerdem eine wirtschaftliche Betrachtung des Projektes durchgeführt.

### 3.1 Ist-Analyse

Um den in diesem Abschnitt beschriebenen Ist-Zustand besser nachvollziehen zu können, befindet sich im Anhang A.3: **Screenshot der Exceltabelle** auf Seite iii ein Screenshot der bislang verwendeten Excel-Datei.

Wie bereits in Abschnitt 1.2 (**Projektbeschreibung**) erwähnt wurde, sind die Fachabteilung **Orga** sowie die Makler der **AO** für die Beratung in der Pflegeversicherung zuständig. Der Sachbearbeiter erfasst das Geburtsjahr des Kunden für die Berechnung des aktuellen Alters. Als nächstes wird erfasst, ob der Tarif *PflegeBahr* berücksichtigt werden soll. Der *PflegeBahr* ist ein staatlich geförderter Tarif, welcher prozentual in jedem Pflegegrad individuell einen Teil der Pflegeücke schließt. Kunden erhalten hier eine staatliche Zulage in Höhe von 5€ im Monat. Der Eigenanteil des Kunden beläuft sich bis zum 40. Lebensjahr auf 10€ pro Monat. Ab einem Eintrittsalter von 41 Jahren steigt der Eigenanteil. Die Auswahl des Tarifs *PflegeBahr* wirkt sich auf die Höhe des möglichen Tagegeldes im Tarif *PflegeSchutz* aus und ist wichtig für die Berechnung des Gesamtbeitrages.

Die Tabelle in der Excel-Datei wird dem Kunden bei der Beratung vorgeführt. Sie soll die Veränderung des Zahlbeitrages visualisieren. Anhand der Eingaben zeigt die Tabelle für die Pflegegrade 1-5 jeweils den berechneten Einzelbeitrag und den insgesamt aufzubringenden Beitrag an. Dieser basiert

### 3 Analysephase

---

neben den genannten Benutzereingaben auf der gewünschten Höhe des Tagegeldes, das der Kunde abschließen möchte.

Durch die Mathematik-Abteilung wird eine Wertetabelle mit den Berechnungsgrundlagen gepflegt. Die Werte ergeben sich aus den Berechnung der Beiträge und müssen zum Jahreswechsel oder zum 01. Juli, im Falle einer Beitragsanpassung ([BAP](#)), aktualisiert werden. Die berechneten Werte werden in einer Datenbank abgelegt. Zusätzlich müssen die Tabellen der Excel-Datei mit den neuen Daten aktualisiert werden.

Bei einer BAP zum Jahreswechsel muss die Datei durch die Mathematik aktualisiert werden. Anschließend muss der neuste Stand an alle Sachbearbeiter, die die Datei verwenden, ausgerollt werden. Eine Überprüfung, ob jeder Sachbearbeiter die veraltete Datei durch die aktualisierte ausgetauscht hat, erfolgt nicht.

Die Excel-Datei beinhaltet außerdem ein Diagramm, das die geschätzten monatlichen Kosten im Pflegefall in Form eines Säulendiagramms für jeden Pflegegrad darstellt. Hierbei wird unterschieden, ob die Pflege stationär in einem Pflegeheim oder zuhause erfolgt. Der Anteil der gesetzlichen Pflegeversicherung und der Eigenanteil werden in der jeweiligen Säule in unterschiedlichen Farben zu festgelegten Beträgen visualisiert.

Wenn ein Kunde nicht persönlich, sondern beispielsweise über das Telefon beraten wird, erstellt der Sachbearbeiter während der Beratung einen Screenshot von der Excel-Datei und schickt diesen per Mail zusammen mit dem Angebot an den Kunden.

## 3.2 Wirtschaftlichkeitsanalyse

Aufgrund des geschilderten Sachverhalts der in Abschnitt [1.4 \(Projektbegründung\)](#) und in Abschnitt [3.1 \(Ist-Analyse\)](#) beschrieben wurde, ist die Umsetzung des Projekts erforderlich. Ob die Realisierung und die damit verbundenen wirtschaftlichen Aufwendungen gerechtfertigt sind, soll in den folgenden Abschnitten betrachtet werden.

### 3.2.1 „Make or Buy“-Entscheidung

Bei dem Projekt geht es unter Anderem um den Vertrieb des Tarifs *PflegeSchutz*. Dieser Tarif ist ein unternehmensspezifisches Produkt der AO. Bei der Recherche und Betrachtung von Software mit vergleichbarer Funktionalität konnte festgestellt werden, dass keine der vorhandenen Anwendungen am Markt die Anforderungen erfüllt und im Verhältnis zu den im nächsten Abschnitt kalkulierten Projektkosten steht. Daher soll das Projekt in Eigenentwicklung durchgeführt werden.

### 3.2.2 Projektkosten

Im Folgenden sollen die Kosten, die während der Entwicklung des Projekts anfallen, kalkuliert werden. Bei der Kalkulation müssen neben den Personalkosten des Entwicklers, der Fachabteilung und anderer Projektbeteiligten auch die Kosten für die verwendeten Ressourcen berücksichtigt werden. Da die Personalkosten nicht herausgegeben werden dürfen, wurde die folgende Kalkulation mit Stundensätzen<sup>3</sup> berechnet, die von der Personalabteilung festgelegt wurden. Der Satz für einen Auszubildenden beträgt 10€, der eines Mitarbeiters aus der Fachabteilung 26€ und der eines Mitarbeiters aus der IT-Abteilung 35€. Für die Ressourcennutzung wird eine Pauschale von 15€ pro Stunde berechnet. Für einen Web-Server, der für den Betrieb der Web-Anwendung notwendig ist, die Wartung und anfallende Lizenzkosten, z. B. für den Application-Server werden jährlich 120€ einkalkuliert.

Es ergeben sich aus den Projektvorgängen und dem Zeitrahmen von 70h für die Entwicklung die in Tabelle 2 dargestellten Projektkosten von 2228€.

Die Kosten des Auszubildenden für die Erstellung des Lastenhefts, der Mockups und die Schulung sind in den Entwicklungskosten enthalten.

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	10 € + 15 € = 25 €	1750 €
Erstellen des Lastenhefts und der Mockups	2 h	26 € + 15 € = 41 €	82 €
Fachgespräch	3 h	26 € + 15 € = 41 €	123 €
Abnahme der Pull-Requests	3 h	35 € + 15 € = 50 €	150 €
Abnahme und Vorführung der Web-Anwendung	3 h	26 € + 15 € = 41 €	123 €
			<b>2228 €</b>

Tabelle 2: Kostenaufstellung

### 3.2.3 Amortisationsdauer

Im Folgenden wird ermittelt, ab wann sich die Umsetzung des Projekts amortisiert hat. Die Anschaffungskosten aus Tabelle 2 werden dabei den Kostenersparnissen, die aus den Zeitersparnissen aus Tabelle 3 resultieren, gegenübergestellt.

Durch das Ablösen der Excel-Datei entfällt manueller Aufwand in der Fachabteilung. So wird die jährliche Aktualisierung der Berechnungsdaten automatisiert. Auch das ausführliche Testen der Funktionalität und das Versionieren nach [IDV-Leitlinie](#) entfällt.

Bei der [AO](#) wird ein Personentag ([PT](#)) grundsätzlich mit 7,6 Stunden definiert, was 456 Minuten entspricht. Wie in Abschnitt 3.1 ([Ist-Analyse](#)) erwähnt, findet eine [BAP](#) immer zum Jahreswechsel und gegebenenfalls zum 01. Juli statt. Deshalb wird die Häufigkeit der aufgeführten Schritte im Durchschnitt auf 1,5 mal im Jahr festgelegt.

<sup>3</sup>Die aufgeführten Stundensätze setzen sich insbesondere aus dem Gehalt und den Sozialaufwendungen des Arbeitgebers zusammen.

### 3 Analysephase

#### Berechnung der Amortisationsdauer

Beim Zeitersparnis von 513 Minuten für den Sachbearbeiter ergibt sich ein Ersparnis von 350,55€ jährlich.

$$\frac{513 \text{ Min/Jahr}}{60} \cdot (26 + 15) \text{ €/h} \approx 350,55 \text{ €/Jahr} \quad (1)$$

Die Amortisationsdauer beträgt, unter Berücksichtigung der 120€ für Wartung und Lizenzen, somit

$$\frac{2.228,00 \text{ €}}{(350,55 - 120,00) \text{ €/Jahr}} \approx 9,66 \text{ Jahre.} \quad (2)$$

Durch die Personalkosteneinsparungen und -verlagerung würde sich das Projekt erst nach 9,66 Jahren amortisieren. Durch die im nächsten Abschnitt behandelten nicht-monetären Aspekte wurde trotz der langen Amortisationsdauer beschlossen, das Projekt umzusetzen. Im Anhang A.4: [Amortisation](#) auf Seite iv ist die Amortisationsdauer grafisch dargestellt.

Vorgang	Anzahl pro Jahr	Zeit (alt) pro Vorgang	Zeit (neu) pro Vorgang	Einsparung pro Jahr
Aktualisieren der Excel-Datei	1,5	228 min (0,5 PT)	-	342 min
Testen und Versionieren nach Änderungen	1,5	114 min (0,5 PT)	-	171 min
				<b>513 min</b>

Tabelle 3: Zeiteinsparung

### 3.3 Nicht-monetäre Aspekte

Die in Abschnitt 1.4 ([Projektbegründung](#)) erwähnte Individuelle Datenverarbeitung (IDV)-Leitlinie definiert, dass die Anzahl der IDV-Anwendungen im Unternehmen reduziert werden soll. Dies ist vom Vorstand festgelegt und bekräftigt die Entscheidung, das Projekt umzusetzen. Ein weiterer Vorteil ist, dass der Mitarbeiter aus der Fachabteilung nicht mehr für die Wartung der Excel-Datei zuständig ist. Hierdurch kann er sich auf das Kerngeschäft der AO konzentrieren. Beim Versand einer E-Mail nach einer telefonischen Beratung wird des Weiteren eine durch die Web-Anwendung erstellte Portable Document Format (PDF)-Datei vom Sachbearbeiter versendet und nicht wie bislang ein manuell erstellter Screenshot der Excel-Datei. Hierdurch soll der allgemeine Eindruck beim Kunden verbessert werden.

### 3.4 Anwendungsfälle

Um die fachlichen Anwendungsfälle zu erfassen und eine grobe Übersicht zu liefern, wurde im Zuge der Analyse ein Use-Case-Diagramm erstellt. Es bildet dabei alle Funktionen, die aus Sicht des Endanwenders benötigt werden, ab. Die beiden Akteure Sachbearbeiter und Makler stehen dabei

in Vererbungsbeziehung. Die Sachbearbeiter decken alle Anwendungsfälle der Makler ab und haben zusätzlich die Möglichkeit, erstellte Vorschläge zu speichern und im Nachhinein wieder aufzurufen. Mit Hilfe des Diagramms konnten Testfälle abgeleitet werden, die bei der Entwicklung nach [TDD](#) notwendig waren. Als das Entity-Relationship-Model ([ERM](#)) erstellt wurde, konnte das Diagramm zusätzlich verwendet werden, um die benötigten Entitätstypen festzustellen. Das Diagramm befindet sich im Anhang [A.5: Use-Case-Diagramm](#) auf Seite [v](#). Der Anwendungsfall *Zahlbeitrag ermitteln* wird in Abschnitt [4.5 \(Geschäftslogik\)](#) näher erläutert.

### 3.5 Lastenheft

Zum Abschluss der Analysephase wurde zusammen mit der Fachabteilung ein Lastenheft erstellt. Zur Priorisierung wurde die MoSCoW-Methode angewendet, bei der die einzelnen Anforderungen mit *muss*, *soll*, *kann* oder *wird nicht* formuliert werden. Mit Hilfe des Lastenhefts konnte festgehalten werden, welche Funktionen für die Fachabteilung am wichtigsten sind, und das Pflichtenheft, das in Abschnitt [4.8 \(Pflichtenheft\)](#) thematisiert wird, abgeleitet werden. Das Lastenheft beinhaltet alle fachlichen Anforderungen und befindet sich im Anhang [A.6: Lastenheft](#) auf Seite [vi](#).

## 4 Entwurfsphase

Nach der Analyse und der Entscheidung das Projekt umzusetzen, wurde ein konkreter, technischer Entwurf der Anwendung erstellt. Außerdem wurde entschieden, welche Technologien sich am Besten für die Umsetzung eignen. Wie dabei vorgegangen wurde, wird in diesem Kapitel beschrieben.

### 4.1 Entwurf der Benutzeroberfläche

Wie in Abschnitt [2.3 \(Entwicklungsprozess\)](#) bereits beschrieben, soll der Fachbereich bei der Entwicklung der Benutzeroberflächen mit eingebunden werden. Hierzu wurden zusammen mit dem Fachbereich Mockups erstellt, die den groben Aufbau und die Anordnung der Elemente auf der Website und in der [PDF](#)-Datei zeigen. Um der Fachabteilung in ihrer Kreativität freien Raum zu lassen und gleichzeitig mögliche Elemente der Oberfläche vorzugeben, wurden Papierschnipsel mit Elementen vorbereitet, die der Mitarbeiter beliebig auf der Oberfläche anordnen konnte. Dadurch konnten die Entwürfe flexibel und schnell erstellt werden, was bei der Fachabteilung sehr gut ankam. Die hierbei entstandenen Mockups werden im Anhang [A.7: Oberflächenentwürfe](#) auf Seite [vii](#) dargestellt. In der Weboberfläche sollen für die Tarife *PflegeSchutz* und *PflegeBahr* jeweils eine Tabelle dargestellt werden. In die [PDF](#)-Datei sollen nur beide Tabellen übernommen werden, wenn auch beide Tarife in der Berechnung berücksichtigt werden sollen. Sollte also beispielsweise nur der *PflegeSchutz* abgeschlossen werden, soll in dem Dokument die Tabelle *PflegeBahr* nicht mit aufgeführt werden.

### 4.2 Zielplattform

Durch das in Abschnitt [1.3 \(Projektziel\)](#) festgelegte Ziel, eine Web-Anwendung zu entwickeln, die im Internet erreichbar sein soll, ergibt sich die Anforderung eines Deployments auf einem Application-Server mit Anbindung an das Internet. Die Datenbank soll mit dem Datenbanksystem *MariaDB*

realisiert werden. Diese ist im Umfeld der Entwicklung von Web-Anwendungen in der AO etabliert. In der IT-Abteilung gibt es genügend Mitarbeiter, die mit diesem System vertraut sind und somit ist die Administration und Wartung sichergestellt.

Die Programmierung wird mit der Sprache Java umgesetzt. Java wird von vielen Entwicklern der AO beherrscht und ist außerdem in der unternehmensinternen Architekturrichtlinie als Standardsprache für die Entwicklung von Web-Anwendungen festgelegt. Dies gilt ebenfalls für die Auswahl des Webframeworks Jakarta Enterprise Edition ([Jakarta EE](#)).

Ausgeführt wird die Web-Anwendung auf der JBoss Enterprise Application Platform ([EAP](#)) 7.2, da die AO Lizenzen für die Nutzung und den Support vom Hersteller besitzt. Des Weiteren kam dieser Application-Server auch bei anderen bereits umgesetzten Projekten zum Einsatz und hat sich dabei als geeignet erwiesen.

### 4.3 Architekturdesign

Das Projekt soll unter Einsatz des Model View Controller ([MVC](#))-Architekturmusters entwickelt werden und durch Komponenten erweitert werden, die im Anhang [A.8: Komponentendiagramm](#) auf Seite [ix](#) dargestellt sind. Grund der Wahl des Architekturdesigns ist die einfache Testbarkeit und Wartung, die sich durch die Trennung der einzelnen Komponenten ergibt.

Die Grundstruktur des [MVC](#)-Musters setzt sich aus drei Schichten zusammen. Jede dieser Schichten hat einen Aufgabenbereich. Die Model-Schicht ist für die Verarbeitung der Daten zuständig, die View-Schicht für die Anzeige der Daten und die Controller-Schicht für die Steuerung der Anwendung.

Die View, und damit die Oberfläche, wird mit dem Java Server Faces ([JSF](#))-Framework des Jakarta-EE-Standards und einer JavaScript-Komponente umgesetzt. Die Controller-Komponente ist die Schnittstelle zur Model-Schicht und bedient sich dafür an den bereits erwähnten Funktionalitäten, die die Services zur Verfügung stellen. Das im Model enthaltene Repository stellt Interfaces zur Verfügung und bildet die Schnittstelle zur Persistenz-Schicht. Die Domäne enthält die Entitäten und die fachliche Logik. Die Services bilden die Use-Cases mit Hilfe der Domänenklassen ab und sind gleichzeitig die Schnittstelle zur Controllerschicht. Im View-Model sind ausschließlich Plain Old Java Object ([POJO](#))-Klassen enthalten, um die Daten aus der Benutzeroberfläche anzunehmen und in die Domäne zu übertragen.

In der Persistenz-Komponente wird eine Schnittstelle zu einer Datenbank geschaffen. Dafür soll in der Anwendung die Java Persistence [API](#) ([JPA](#)) verwendet werden, um Funktionalitäten für das Persistieren und das Auslesen der Daten bereitzustellen. Diese ist ebenfalls ein Standard von [Jakarta EE](#).



## 4.4 Datenmodell

Um das Datenmodell in Form eines [ERMs](#) zu erstellen, wurden das Lastenheft und die einzelnen Anwendungsfälle aus Abschnitt [3.4 \(Anwendungsfälle\)](#) zur Hilfe genommen. Dadurch konnten die Nutzung der Anwendung betrachtet und die Entitätstypen und Attribute ermittelt werden. Bei der Analyse wurde festgestellt, dass die Entitätstypen **Pflegegrad**, **Tarif**, **Vorschlag** und **Benutzer** notwendig sind.

Der **Benutzer**, der in diesem Fall der Makler oder Sachbearbeiter ist, loggt sich mit einem **Benutzernamen** und einem **Passwort** im Pflegerechner ein. Um einen **Vorschlag** zu erstellen, muss er das **Geburtsjahr** des Kunden und die **Eigenleistung** eingeben, sowie die **Tarife**, die abgeschlossen werden sollen, auswählen. Für jeden der Tarife wird ein **Tagegeld** für jeden **Pflegegrad** vereinbart. Die **Bezeichnungen** der einzelnen Grade werden als römische Ziffer dargestellt. Abhängig vom Kundenalter wird dann für jeden Grad der **Zahlbeitrag** ermittelt. Da ein Beitrag auch für mehrere Altersklassen gültig sein kann, muss zu den Beiträgen auch das **Alter von** und das **Alter bis** in der Datenbank abgelegt werden. Aus dem **Vorschlag** kann dann anschließend eine [PDF](#)-Datei erzeugt und mit einem **Dateinamen** abgelegt werden.

Im Nachgang wurde überlegt, ob wichtige Attribute ergänzt werden müssen. Es ist beispielsweise erforderlich, ein **Salt** in der Datenbank abzulegen. Dieser wird einem Passwort vor dem Hashen angehängen, um zu verhindern, dass aus identischen Passwörtern der gleiche Hash erzeugt wird. Das [ERM](#) wird im Anhang [A.9: Entity-Relationship-Model](#) auf Seite [x](#) dargestellt.

Für die Anwendung soll eine relationale Datenbank verwendet werden. Deshalb wurde das konzeptionelle [ERM](#) in ein Tabellenmodell überführt und die Beziehungen in Zwischentabellen aufgelöst. Das entsprechende Diagramm befindet sich im Anhang [A.10: Tabellenmodell](#) auf Seite [xi](#).

## 4.5 Geschäftslogik

Einen der wichtigsten Teile der Geschäftslogik stellt der Use-Case *Zahlbeitrag ermitteln* dar. Es müssen die einzelnen Zahlbeiträge unter Berücksichtigung der Tarife ermittelt werden. Dafür muss zunächst das Alter des Kunden durch Eingabe in die Benutzeroberfläche erfasst werden, ob der Tarif *PflegeBahr* berücksichtigt werden soll und ob der Kunde eine Eigenleistung im Pflegefall leisten möchte. Mit dem Alter wird der anfallende Beitrag pro 5€ Tagegeld für jeden Pflegegrad in den beiden Tarifen ermittelt. Anhand der geschätzten Pflegekosten werden abzüglich der Leistungen der gesetzlichen Pflegeversicherung, der Leistungen des *PflegeBahr*, falls ausgewählt, und des Eigenanteils die Pflegelücken in den Pflegegraden für die stationäre und ambulante Pflege ermittelt. Durch die Eingabe von Tagegeldern, die der Kunde in den einzelnen Pflegegraden versichern möchte, sowie dem Beitrag aus dem *PflegeBahr* werden die einzelnen Zahlbeiträge des *PflegeSchutz* und der Gesamtbeitrag berechnet. Um diesen komplexeren Use-Case besser nachvollziehen zu können, wurde ein Aktivitätsdiagramm erstellt. Das Diagramm befindet sich im Anhang [A.11: Aktivitätsdiagramm](#) auf Seite [xii](#).



## 4.6 Maßnahmen zur Qualitätssicherung

In Abschnitt 2.3 ([Entwicklungsprozess](#)) wurde bereits thematisiert, dass die Praktik des TDD zum Einsatz kommen soll. Durch das Entwickeln der Tests vor der eigentlichen Implementierung des Quellcodes werden fachliche Fehler von Anfang an vermieden. Des Weiteren können die Tests, z. B. nach einer Refaktorisierung nach der Pfadfinderregel<sup>4</sup>, ausgeführt werden um die Funktionalität zu kontrollieren.

Durch den Prozess der Branches `master`, `qs` und der einzelnen Branches für jedes Feature, das entwickelt wird, ergeben sich weitere Maßnahmen, die die Qualität des Projekts steigern. Durch die Abnahme der PRs und die Tests der Fachabteilung wird bereits während der Entwicklung sichergestellt, dass sowohl die Fachlichkeit als auch die Qualität des Codes den Anforderungen entsprechen. Unterstützend kommt das Tool *SonarQube* zum Einsatz. Es beinhaltet verschiedene Prüfungen, die die Qualität des Codes nach jedem Build beurteilen. Hierzu gehören die Suche nach doppeltem Code und Code-Smells und eine Testabdeckung von mindestens 80% des neu hinzugekommenen Codes. Ein Screenshot der Übersicht aus *SonarCube* befindet sich im Anhang A.12: [Testabdeckung Ansicht aus SonarQube](#) auf Seite [xiii](#).

## 4.7 Deployment

Da es sich um eine Web-Anwendung handelt, ist ein Deployment auf einem Web-Server erforderlich. Im Anhang A.13: [Verteilungsdiagramm](#) auf Seite [xiv](#) wird dargestellt, welche Systeme am Deployment beteiligt sind.

Als Entwicklungsumgebung kommt die *Eclipse IDE for Enterprise Java Developers* zum Einsatz. Gebaut wird das Projekt mit *Gradle*. *Gradle* ist ein Build-Management-Tool, mit dem Anwendungen durch einen automatisierten Prozess erzeugt werden können. Benötigte Abhängigkeiten werden hierbei über ein Binary-Repository geladen. Der Vorteil des automatisierten Build-Prozesses ist, dass jeder Entwickler, der die Anwendung in Zukunft warten wird, keinen manuellen Konfigurationsaufwand mehr hat.

Zu Testzwecken wird die Web-Anwendung während der Implementierungsphase laufend auf einer JBoss-Instanz in *Docker* deployt. Die Datenhaltung erfolgt über eine Testdatenbank in Form einer *MariaDB*, die ebenfalls in einem lokalen Docker-Container läuft. Die Docker-Container und das notwendige Docker-Image werden über eine YAML Aint't Markup Language ([YAML](#))-Datei bzw. eine Build-Datei erstellt.

Wie in Abschnitt 2.3 ([Entwicklungsprozess](#)) beschrieben, wird auf den Branch `master` deployt, wenn ein PR vom Branch `qs` in den `master` alle Voraussetzungen erfüllt, von einem Mitarbeiter aus der IT gereviewt wurde und der Autor die Branches zusammengeführt hat. Hierdurch wird über einen angelegten Job der `master` automatisiert mit *Jenkins* in einem Docker-Container gebaut. Über das *Jenkinsfile*, welches sich im Anhang A.14: [Jenkinsfile \(Auszug\)](#) auf Seite [xv](#) befindet, ist konfiguriert, dass beim Bauen von `qs` und `master` zwei zusätzlichen Schritte, *Publish Image* und

---

<sup>4</sup>Vgl. [ROBERT C. MARTIN \[2008\]](#).

*Deployment*, ausgeführt werden. Im Schritt *Publish Image* wird ein Docker-Image erstellt und in das Binary-Repository *Artifactory* gepusht. Im Schritt *Deployment* stellt *Jenkins* eine Verbindung zum Webserver *QS* oder *Prod* her und pullt das neue Docker-Image. Aus dem Image wird in *Docker* ein neuer Container hochgefahren, in dem ein *JBoss-Application-Server* betrieben wird. Der Application-Server nutzt eine *MariaDB*. Die Verbindung zu der *MariaDB* wird hergestellt, wenn eine sogenannte Datasource in der JBoss-Konfiguration eingerichtet wurde. Das Skript, das für das Einrichten dieser Datasource verwendet wird, befindet sich im Anhang [A.15: JBoss Konfiguration \(Auszug\)](#) auf Seite [xvi](#). Nach dem Deployment kann der Client über einen Webbrowser die Web-Anwendung anfragen.

## 4.8 Pflichtenheft

Am Ende der Entwurfsphase wurde das Pflichtenheft erstellt. Die Anforderungen, die im Lastenheft aus Abschnitt [3.5 \(Lastenheft\)](#) aufgenommen wurden, bilden die Grundlage für das Pflichtenheft. Das Pflichtenheft dient als Leitfaden bei der Realisierung des Projektes. Das Pflichtenheft befindet sich im Anhang [A.16: Pflichtenheft](#) auf Seite [xvii](#).

# 5 Implementierungsphase

Wie in Abschnitt [2.3 \(Entwicklungsprozess\)](#) beschrieben, erfolgte die Implementierung inkrementell in Form von einzelnen Features. So wurden für jedes Feature zuerst die Oberfläche und dann beginnend mit den Tests die benötigten Klassen bis hin zur Persistenzschicht implementiert.

## 5.1 Implementierung der Benutzeroberfläche

Mit Hilfe der erstellten Mockups aus Kapitel [4.1 \(Entwurf der Benutzeroberfläche\)](#) wurde die Implementierung der Benutzeroberfläche des ersten Features begonnen. Wie in Kapitel [4.3 \(Architekturdesign\)](#) beschrieben, wurde dabei das Framework **JSF** verwendet. Mittels der Expression Language aus **JSF** können Controller aufgerufen werden. Für die Darstellung im Browser wird daraus Hypertext Markup Language (**HTML**) gerendert.

Die Hauptseite wurde aus mehreren Extensible Hypertext Markup Language (**XHTML**)-Dateien zusammengesetzt. Beispielsweise wurden die Diagramme in einer eigenen Datei gekapselt. Dadurch bleiben die einzelnen Dateien übersichtlich. Für die Tabelle mit den Tagelgeldern im Tarif *PflegeBahr* wurde eine Checkbox auf der Hauptseite implementiert. Wird die Checkbox ausgewählt, wird über die Expression Language das Attribut `pflegebahr` auf `true` gesetzt und ein Asynchronous JavaScript and XML (**AJAX**)-Event ausgelöst. Das Event aktualisiert den Container `pflegebahrtabelle`, in dem das Attribut mit einer `if`-Anweisung abgefragt wird und das anschließend die Datei mit der PflegeBahr-Tabelle lädt. Der Code der Hauptseite befindet sich im Anhang [A.17: Listing von JSF-Code](#) auf Seite [xviii](#).

Für die Säulendiagramme wurden mehrere JavaScript Funktionen implementiert. Ziel war es, dass sich das Diagramm direkt aktualisiert, wenn sich dafür relevante Daten wie zum Beispiel die Eigenleistung oder die Höhe des Tagelgeldes ändern. Es wurde eine Funktion implementiert, die beim

Aufruf der Seite ausgeführt wird und `EventListener` registriert. Ein Auszug des Scripts befindet sich im Anhang [A.18: Listing von JavaScript-Code](#) auf Seite [xix](#).

Ein Screenshot der fertigen Anwendung befindet sich im Anhang [A.19: Screenshots der fertigen Anwendung](#) auf Seite [xx](#)

## 5.2 Implementierung der Datenstrukturen

Um die Domänenklassen zu implementieren, die für die Datenhaltung aus der Persistenz-Schicht zuständig sind, wurde das [ERM](#) im Anhang [A.9: Entity-Relationship-Model](#) auf Seite [x](#) als Grundlage verwendet. Hieraus ließen sich alle benötigten Klassen aus den Entitätstypen ableiten und die jeweiligen Attribute übertragen. Um die Domänenklassen mit den Tabellen der Datenbank zu verknüpfen, wurden die Klassen mit [JPA](#)-Annotationen versehen. Da die Namen der Tabellen mit den Klassennamen übereinstimmen, reichte es aus, die Klassen mit der Annotation `@Entity` zu kennzeichnen. Das Primärschlüsselattribut musste mit `@Id` festgelegt werden. Die Kardinalitäten wurden über Attribute in den entsprechenden Klassen und Annotationen wie beispielsweise `@ManyToOne` und `@JoinColumn(name = "id")` realisiert. So besitzt beispielsweise die Klasse `Benutzer` ein Attribut vom Datentyp `List<Vorschlag>` mit den Annotationen `@OneToMany` und `@JoinColumn(name = "id")`. Die Klasse `Vorschlag` besitzt wiederum ein Attribut vom Typ `Benutzer` mit den Annotationen `@ManyToOne` und `@JoinColumn(name = "id")`.

Um die Berechnungsdaten in die MariaDB zu importieren, wird eine Representational State Transfer ([REST](#))-Schnittstelle implementiert, die über die HTTP-Request-Methode `POST` die Daten als JavaScript Object Notation ([JSON](#)) annimmt. Diese Daten werden dann von der Persistenz-Schicht persistiert.

## 5.3 Implementierung der Geschäftslogik

Bei der Implementierung der Service-Klassen mit der Geschäftslogik wurde unter Anderem das Use-Case-Diagramm aus Anhang [A.5: Use-Case-Diagramm](#) auf Seite [v](#) zur Hilfe genommen. Außerdem war das Aktivitätsdiagramm, welches sich im Anhang [A.11: Aktivitätsdiagramm](#) auf Seite [xii](#) befindet hilfreich. Die Aktivitäten aus dem Diagramm dienten als Leitfaden, um alle benötigten Funktionalitäten der Service-Schicht abzubilden. Um das Konzept [TDD](#) einzuhalten, wurde vor der Implementierung eines Use-Cases zunächst ein Testfall erstellt. Da für die Methoden der Services Daten aus der Persistenz-Schicht vorhanden sein müssen, wurden die Repositorys mit dem Framework Mockito „gestubt“. Es wurden dabei sogenannte Stub-Objekte erstellt, die genau das Verhalten der Persistenz-Objekte besitzen, jedoch festgelegte Werte zurückgeben anstatt auf die Datenbank zuzugreifen. Die Service-Klassen erhalten als Schnittstelle zur Persistenz Interfaces als Instanzvariablen. Der Konstruktor wird mit der Annotation `@Inject` gekennzeichnet. Dadurch werden die Parameter automatisch durch Contexts and Dependency Injection ([CDI](#)) instanziiert. Die Interfaces werden jeweils von einer Klasse implementiert und die Schnittstelle zur Datenbank ist somit leicht austauschbar. Wenn in Zukunft die Datenbank ausgetauscht werden soll, können neue Klassen entwickelt werden, die die Interfaces implementieren. Wichtiger Teil der Geschäftslogik ist das Berechnen der Pflegelücke. Es musste berücksichtigt werden, ob es sich um die Lücke in der ambulanten oder

der stationären Pflege handelt. Dies hat Auswirkungen darauf, ob der Entlastungsbeitrag vom Staat gezahlt wird oder nicht. Des Weiteren ist wichtig, ob der *PflegeBahr* ausgewählt wurde und die Leistungen gegebenenfalls aus der Datenbank ermittelt werden. Um die Pflegelücke abschließend zu berechnen, muss außerdem der in der Benutzeroberfläche angegebene Tagesgeldsatz mit einbezogen werden. Außerdem muss die Pflegelücke dem Controller in Form eines Datentransferobjekt (DTO) zurückgegeben werden. Im Anhang [A.20: Listing von Java-Code](#) auf Seite [xxi](#) befinden sich Auszüge aus den entsprechenden Klassen und Testklassen, die unter Anderem die beschriebene Funktionalität enthalten.

## 5.4 Testen der Anwendung

Die Controller- und Serviceklassen wurden mit JUnit-Tests abgedeckt. Das Verhalten der Datenbank wurde wie im vorherigen Abschnitt erwähnt mittels des Frameworks Mockito simuliert. Für die Oberfläche wurden Tests mit dem Framework *Arquillian* erstellt. *Arquillian* lässt dabei die Anwendung in einem Headless-Browser laufen. Um das Verhalten der Repositorys zu testen, wurden Integrationstests entwickelt. Beim Build wird dabei ein eigener Docker-Container mit einer MariaDB hochgefahren und mit Testdaten gefüllt. Die Tests arbeiten also mit einer echten Datenbank, die nach der Ausführung der Tests wieder automatisch gelöscht wird. Durch die Trennung der JUnit-Tests und der Tests der Repositorys ergibt sich der Vorteil, dass bei einem Problem mit der Datenbank die restlichen Tests davon abgekapselt trotzdem erfolgreich ausgeführt werden können. Das kann sich positiv auf die Fehleranalyse auswirken. Die daraus resultierende Testabdeckung lässt sich aus der *SonarQube* Oberfläche im Anhang [A.12: Testabdeckung Ansicht aus SonarQube](#) auf Seite [xiii](#) entnehmen.

## 6 Abnahme- und Einführungsphase

Wie in Abschnitt [4.7 \(Deployment\)](#) beschrieben, wurde das Deployment weitestgehend automatisiert. Im Zuge der Einführungsphase wurde kontrolliert, ob das Deployment auf der produktiven Umgebung erfolgreich durchgeführt wurde.

Eine Schulung der Mitarbeiter war durch den ständigen Austausch während des Projekts nicht notwendig. Die fertige Anwendung wurde der Fachabteilung im Rahmen der Einführungsphase in der dafür vorgesehenen Zeit vorgeführt. Parallel wurde die Anwendung durch die Fachabteilung in verschiedenen Browsern und auf einem Tablet getestet. Dies hatte den Vorteil, dass abschließend nochmal auf potenzielle Fehler geachtet und die Anwender gleichzeitig mit der Bedienung vertraut gemacht wurden.

Die Anwendung wurde nach der Vorführung durch die Fachabteilung abgenommen.

## 7 Dokumentation

Die Dokumentation setzt sich aus dem Benutzerhandbuch und der Entwicklerdokumentation zusammen. Das Benutzerhandbuch dient dazu, die Makler in die Nutzung der Oberfläche einzuführen. Ein

Ausschnitt des Benutzerhandbuchs befindet sich im Anhang [A.21: Benutzerdokumentation \(Auszug\)](#) auf Seite [xxiii](#). Die Entwicklerdokumentation dient dazu, Entwicklern einen Überblick über die Webanwendung zu geben und so die Einarbeitung zu beschleunigen. Sie soll die einzelnen Klassen sowie deren Attribute und Methoden erläutern. Die Dokumentation wurde in Form von *JavaDoc* erstellt. Dadurch konnte mithilfe der Entwicklungsumgebung ein [HTML-Dokument](#) generiert werden. Ein Ausschnitt der Entwicklerdokumentation befindet sich im Anhang [A.22: Entwicklerdokumentation \(Auszug\)](#) auf Seite [xxiv](#). Für einen Überblick über die Klassen und deren Beziehungen wurde ein Klassendiagramm generiert. Ein Auszug davon befindet sich im Anhang [A.23: Klassendiagramm der Domäne \(Auszug\)](#) auf Seite [xxv](#).

## 8 Fazit

Abschließend lässt sich sagen, dass das Projekt auch ohne großen wirtschaftlichen Vorteil für das Unternehmen ein Erfolg war. Der Aufwand bei der Verwaltung der dezentralen Anwendung konnte reduziert werden. Außerdem geht die Fachabteilung davon aus, dass sich die Qualität der Beratung verbessert und sich positiv auf die Abschlusszahlen auswirken könnte.

### 8.1 Soll-/Ist-Vergleich

Der zu Beginn des Projektes erstellte Projektplan aus Abschnitt [2.1 \(Projektphasen\)](#) wurde eingehalten. In der Implementierungsphase ergaben sich kleinere Abweichungen innerhalb der einzelnen Punkte. Bei der Implementierung der Datenbankanbindung konnte eine Stunde eingespart werden. Die Implementierung des Säulendiagramms hat dagegen ca. eine Stunde mehr in Anspruch genommen, sodass sich diese Zeitabweichungen aufheben.

### 8.2 Lessons Learned

Im Zuge der Umsetzung konnten wertvolle Erfahrungen bezüglich Planung und Durchführung von Projekten gesammelt werden. Es wurde deutlich, dass die Kommunikation zur Fachabteilung während der Planungs- und Entwurfsphase wichtig für die erfolgreiche Umsetzung des Projekts ist. Im Bezug zur Programmierung konnte der Autor wichtige Grundlagen, wie zum Beispiel das Anwenden von [TDD](#), vertiefen. Durch die Verwendung von [Jakarta EE](#) konnten Fähigkeiten im Bezug auf die Webentwicklung und die Verwendung von verschiedenen Frameworks wie zum Beispiel [JSF](#) und [JPA](#) gewonnen werden.

### 8.3 Ausblick

Alle im Pflichtenheft definierten Anforderungen wurden während der Umsetzung des Projekts erfüllt. Trotzdem ist nicht auszuschließen, dass die Anwendung in Zukunft um weitere Tarife wie zum Beispiel einen Krankentagegeld-Tarif erweitert werden könnte. Um die Nutzung für Makler zu vereinfachen, könnte das Benutzerhandbuch durch Tooltips an den jeweiligen Elementen ersetzt bzw. ergänzt werden. Denkbar wäre außerdem, dass das Schwesterunternehmen der [AO](#), die Provinzial Krankenversicherung, ebenfalls an der Webanwendung interessiert ist.

## Literaturverzeichnis

### ALTE OLDENBURGER 2019

ALTE OLDENBURGER: *Geschäftsbericht 2019 ALTE OLDENBURGER Krankenversicherung AG.* [https://www.alte-oldenburger.de/files/medien/Ueber\\_Uns/Geschaeftsberichte/Geschaeftsbericht2018ALTE.OLDENBURGER.Krankenversicherung.AG.pdf](https://www.alte-oldenburger.de/files/medien/Ueber_Uns/Geschaeftsberichte/Geschaeftsbericht2018ALTE.OLDENBURGER.Krankenversicherung.AG.pdf). Version: 2019, Abruf: 02.05.2020

### Bundesministerium der Justiz und für Verbraucherschutz 2019

BUNDESMINISTERIUM DER JUSTIZ UND FÜR VERBRAUCHERSCHUTZ: *Sozialgesetzbuch (SGB) - Elftes Buch (XI) - Soziale Pflegeversicherung (§ 14 SGB XI Begriff der Pflegebedürftigkeit).* <https://www.sozialgesetzbuch-sgb.de/sgbxi/14.html>. Version: 2019, Abruf: 02.05.2020

### Robert C. Martin 2008

ROBERT C. MARTIN: *Clean Code – A Handbook of Agile Software Craftmanship.* August 2008

## A Anhang

### A.1 Detaillierte Zeitplanung

<b>Analysephase</b>	<b>7 h</b>
1. Durchführen der Ist-Analyse	2 h
2. Durchführen der Wirtschaftlichkeitsanalyse	3 h
3. Ermitteln von Use-Cases und Erstellen eines Anwendungsfall-Diagramms	1 h
4. Unterstützen des Fachbereiches beim Erstellen des Lastenheftes	1 h
<b>Entwurfsphase</b>	<b>11 h</b>
1. Entwerfen der Benutzeroberflächen	3 h
2. Entwerfen der Datenbankstruktur und eines ER-Modells	2 h
3. Ableiten des Tabellen- und Domänenmodells aus dem ER-Modell	2 h
4. Planen der Architektur inkl. Erstellen eines Komponentendiagramms	1 h
5. Erstellen des Pflichtenheftes	3 h
<b>Implementierungsphase inkl. Tests</b>	<b>42 h</b>
1. Anlegen des Java-Projekts	1 h
2. Einrichten des Jenkins- und Java-Builds sowie der statischen Code-Analyse	2 h
3. Anlegen eines Skripts zum Auslesen der Berechnungsgrundlagen	2 h
4. Implementieren der Domänenklassen inkl. Tests	6 h
5. Implementierung des Datenimports der Berechnungsgrundlagen	5 h
6. Implementieren der Logik zur Berechnung eines Beitrags	6 h
7. Implementierung der Datenbankbindung inkl. Tests	6 h
8. Implementieren der Oberfläche inkl. Tests	7 h
9. Implementieren des Säulendiagramms	7 h
<b>Abnahme und Einführung</b>	<b>5 h</b>
1. Abnahme durch die Fachabteilung	1 h
2. Vorführen der Anwendung in der Fachabteilung	1 h
3. Erfolgskontrolle in der Fachabteilung	1 h
4. Deployment der Anwendung	2 h
<b>Dokumentation</b>	<b>5 h</b>
1. Erstellen des Benutzerhandbuchs	3 h
2. Erstellen der Entwicklerdokumentation	2 h
<b>Gesamt</b>	<b>70 h</b>



## A.2 Ressourcenplanung

### Hardware

- Büroarbeitsplatz mit Thin-Client

### Software

- Windows 10 Enterprise – Betriebssystem
- Eclipse IDE for Enterprise Java Developers 2019-06 – Entwicklungsumgebung für [Jakarta EE](#)
- MariaDB – Datenbanksystem
- Eclipse - Entwicklungsumgebung von  $\text{\LaTeX}$
- MiKTeX – Distribution des Textsatzsystems  $\text{\LaTeX}$
- SonarQube – Plattform für statische Code-Analyse
- Jenkins – Buildserver für Continuous Deployment ([CD](#))
- JUnit – Unit-Test-Framework zum Testen
- Mockito – Mocking-Framework zur Erstellung von Fake-Klassen
- Arquillian – Framework zum Testen von Oberflächen
- JBoss [EAP](#) 7.2 – Application Server
- Docker – Software für das Ausführen von Anwendungen in Containern
- Gradle – Build-Tool
- Gitea – Verteilte Versionsverwaltung
- Redmine – Projektmanagementsoftware
- Enterprise Architect – Programm zum Erstellen von Diagrammen und Modellen
- MySQL Workbench – Verwaltungswerkzeug für MySQL-Datenbanken

### Personal

- Auszubildender Anwendungsentwickler – Umsetzung des Projektes
- Anwendungsentwickler – Review der Pull-Requests
- Mitarbeiter der Fachabteilung – Festlegung der Anforderungen, Gestaltung der Oberflächen und Abnahme der Anwendung



## A.3 Screenshot der Exceltabelle

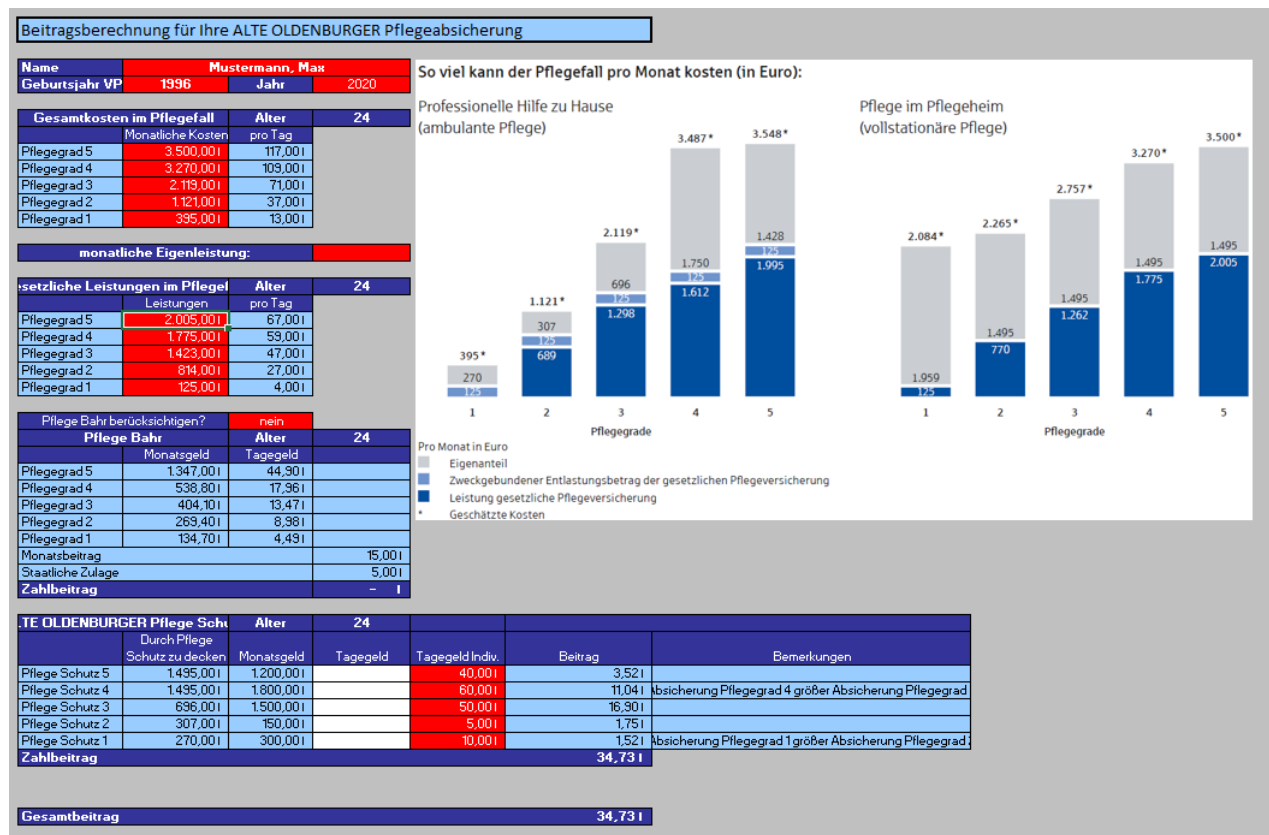


Abbildung 1: Screenshot der Exceltabelle

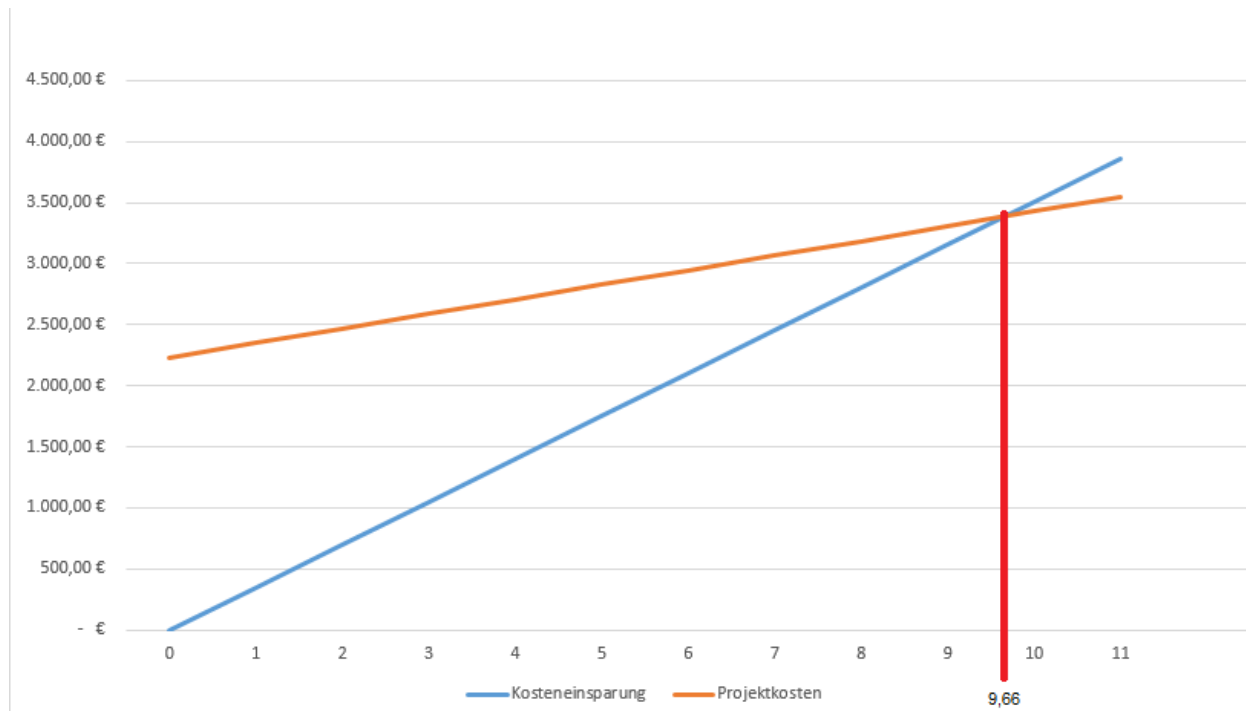
**A.4 Amortisation**

Abbildung 2: Amortisation

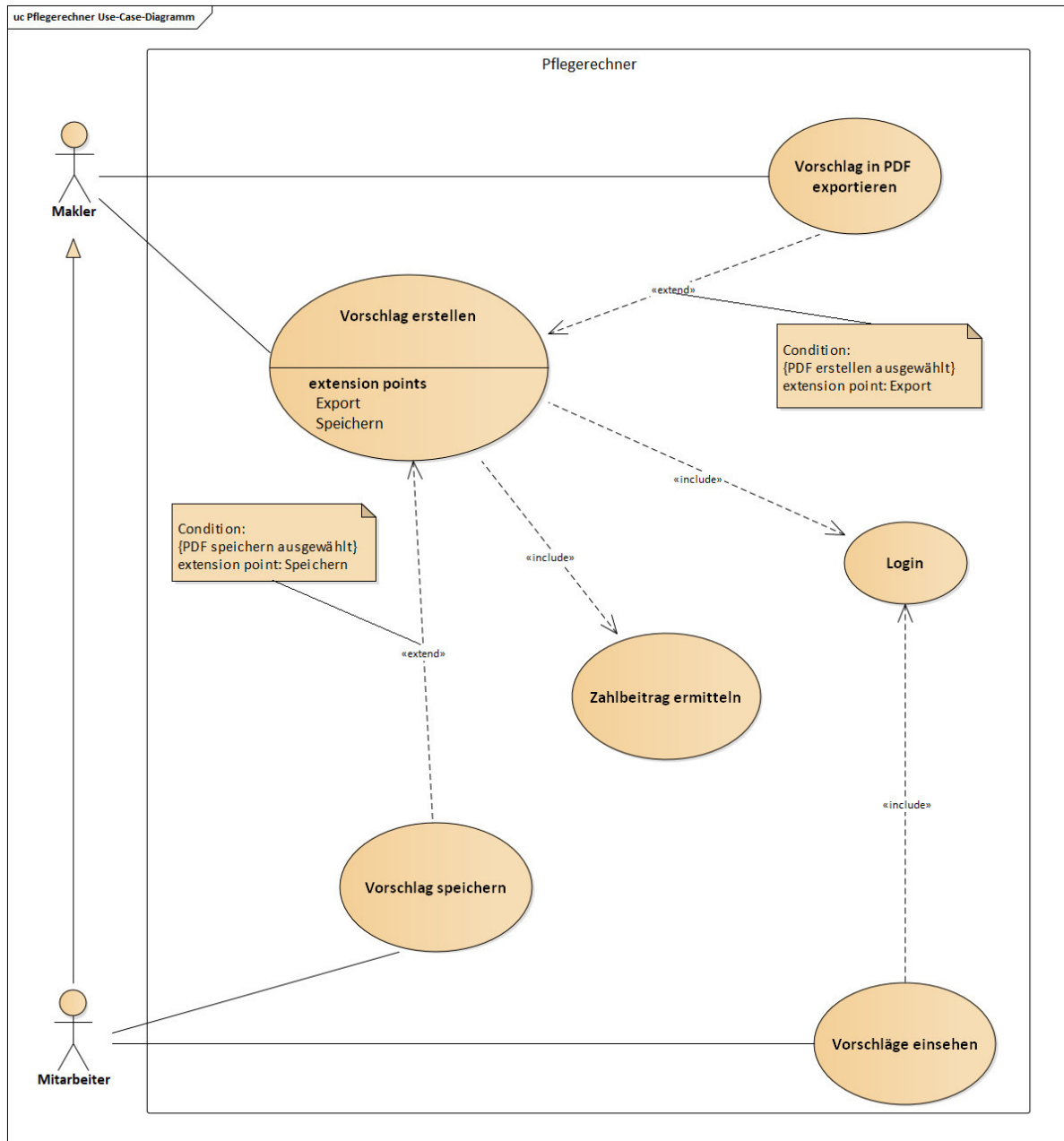
**A.5 Use-Case-Diagramm**

Abbildung 3: Use-Case-Diagramm

## A.6 Lastenheft

Die Anwendung muss folgende Anforderungen erfüllen:

### 1. Verarbeitung der Daten

- 1.1. Es müssen die Berechnungsgrundlagen aus der Mathematik-Abteilung von der IT-Abteilung eingepflegt werden und jährlich nach einer Beitragsanpassung aktualisiert werden.
- 1.2. Die Anwendung muss die Berechnungsgrundlagen bezüglich der Pflegezusatzversicherung auslesen und verarbeiten.
- 1.3. Der Nutzer muss das Geburtsdatum des Kunden eingeben und optional die Höhe der Eigenleistung.
- 1.4. Die Anwendung muss das versicherungstechnische Jahr anhand des Geburtsdatums berechnen.
- 1.5. Der Nutzer muss angeben, ob eine Berücksichtigung des Tarifs *PflegeBahr* erfolgen soll.
- 1.6. Der monatliche Zahlbeitrag sowie die Absicherung im Falle einer Pflegebedürftigkeit müssen anhand der Stamm- und Eingabedaten für jeden Pflegegrad errechnet werden.
- 1.7. Die Anwendung muss verhindern, dass das vereinbarte Pflegeitagegeld höher ist als das Pflegeitagegeld nach einem höheren Pflegegrad.

### 2. Darstellung der Daten

- 2.1. Die Anwendung muss die berechneten Zahlbeiträge und die Absicherung des Tarifs *PflegeSchutz* gestaffelt nach Pflegegrad anzeigen.
- 2.2. Die Anwendung muss die gesetzlichen Leistungen im Pflegefall gestaffelt nach Pflegegrad anzeigen.
- 2.3. Die Anwendung muss die Monats- und Tagegelder des Tarifs *PflegeBahr* gestaffelt nach Pflegegrad anzeigen.
- 2.4. Die Anwendung muss die Zusammensetzung aus *PflegeSchutz*, *PflegeBahr*, Eigenanteil, Entlastungsbetrag und den geschätzten Pflegekosten in einem Säulendiagramm gestaffelt nach Pflegegrad darstellen.

### 3. Sonstige Anforderungen

- 3.1. Die Anwendung muss ohne das Installieren einer zusätzlichen Software über einen Webbrowser im Internet erreichbar sein.
- 3.2. Die Anwendung soll auf mobilen Geräten nutzbar sein.
- 3.3. Die Dialogprinzipien bei der Gestaltung von interaktiver Software sollen eingehalten werden.
- 3.4. Da die Anwendung bei der Beratung zu einem Abschluss in der Pflegezusatzversicherung eingesetzt wird, müssen die angezeigten Daten korrekt sein.
- 3.5. Die Anwendung muss einen Export der Daten inkl. Diagramm als [PDF](#) anbieten.
- 3.6. Die Anwendung soll den Export und die Ergebnisdaten speichern.
- 3.7. Die Anwendung muss durch einen Login abgesichert sein.

## A.7 Oberflächenentwürfe

https://pflegerechner.alte-oldenburger.de

Logo

Eigenleistung  Geburtsjahr

Pflegeschutz

Pflegegrad	Tagegeld	Monatlich	Lü. ambulant	Lü. stationär
V	50,00 €	1500 €	....	....
IV	40,00 €	1200 €	....	....
III	30,00 €	900 €	....	....
II	20,00 €	600 €	....	....
I	10,00 €	300 €	....	....

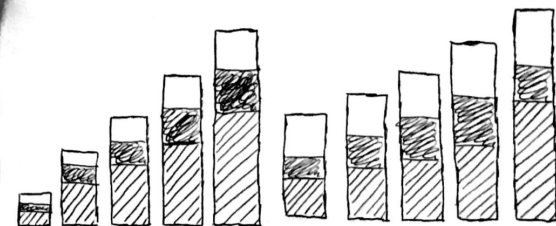
☒ Pflegebahr

Pflegebahr

Pflegegrad	Monatlich
V	1400 €
IV	1100 €
III	800 €
II	400 €
I	100 €

PDF erzeugen

Ambulant Stationär



Legende

Gesamtübersicht

Pflegegrad	Pflegegrad	Betrag
PflegeSchutz	V	0,76 €
"	IV	7,00 €
"	III	10,14 €
"	II	6,44 €
"	I	3,52 €
PflegeBahr	I - V	10,00 €
Gesamt		37,86 €

Abbildung 4: Anzeige der Benutzeroberfläche im Browser

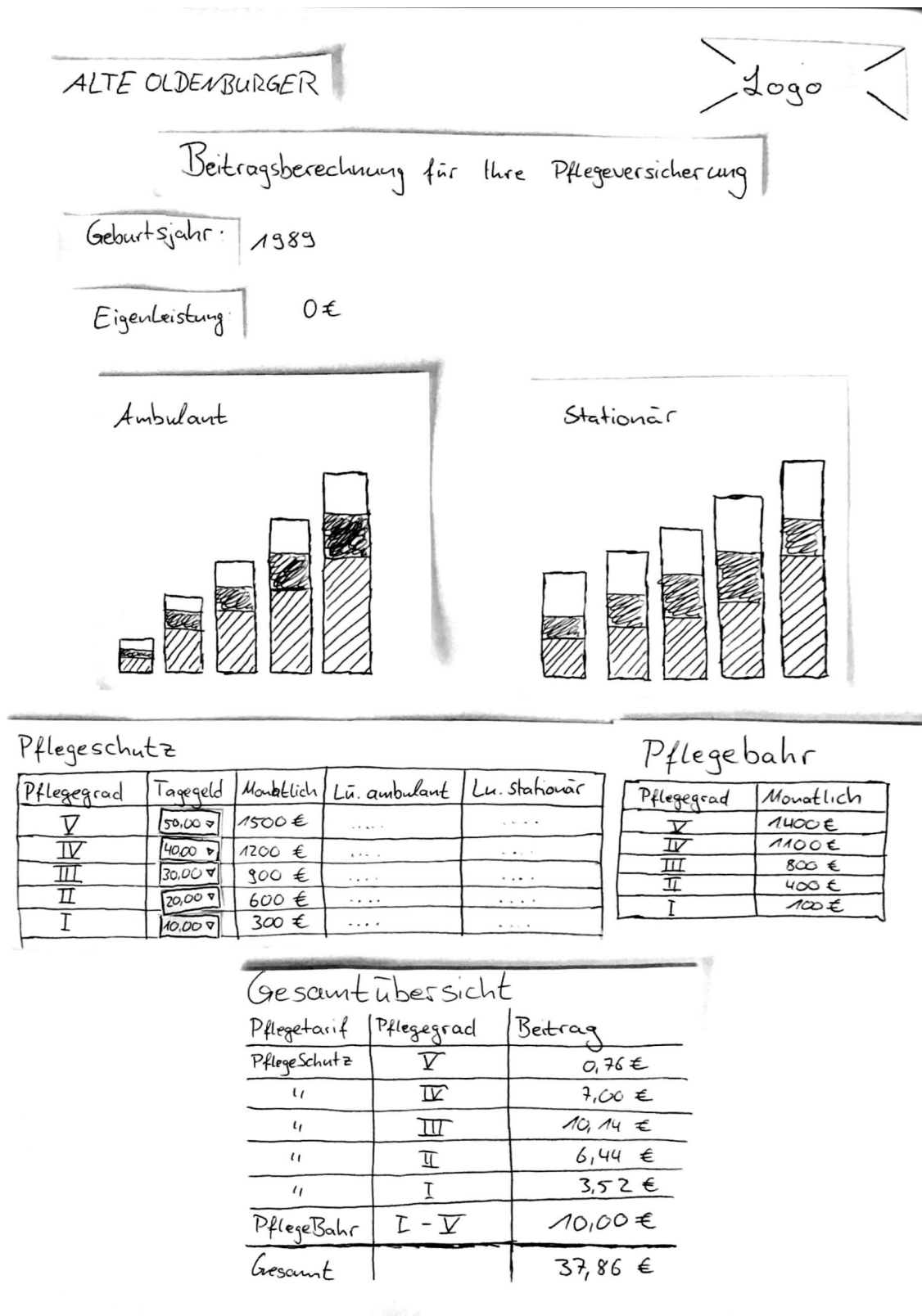


Abbildung 5: Ansicht der generierten PDF-Datei

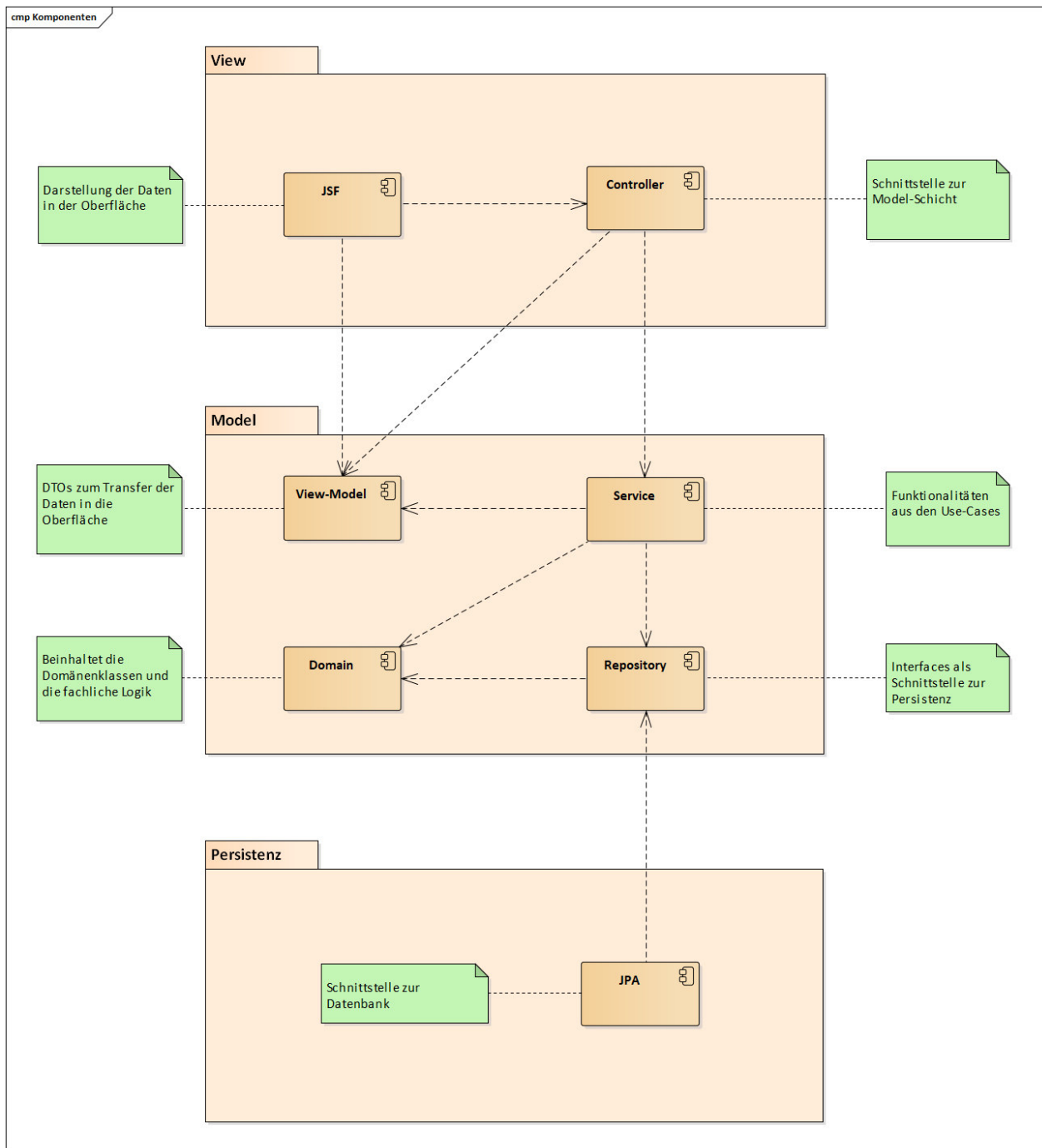
**A.8 Komponentendiagramm**

Abbildung 6: Komponentendiagramm

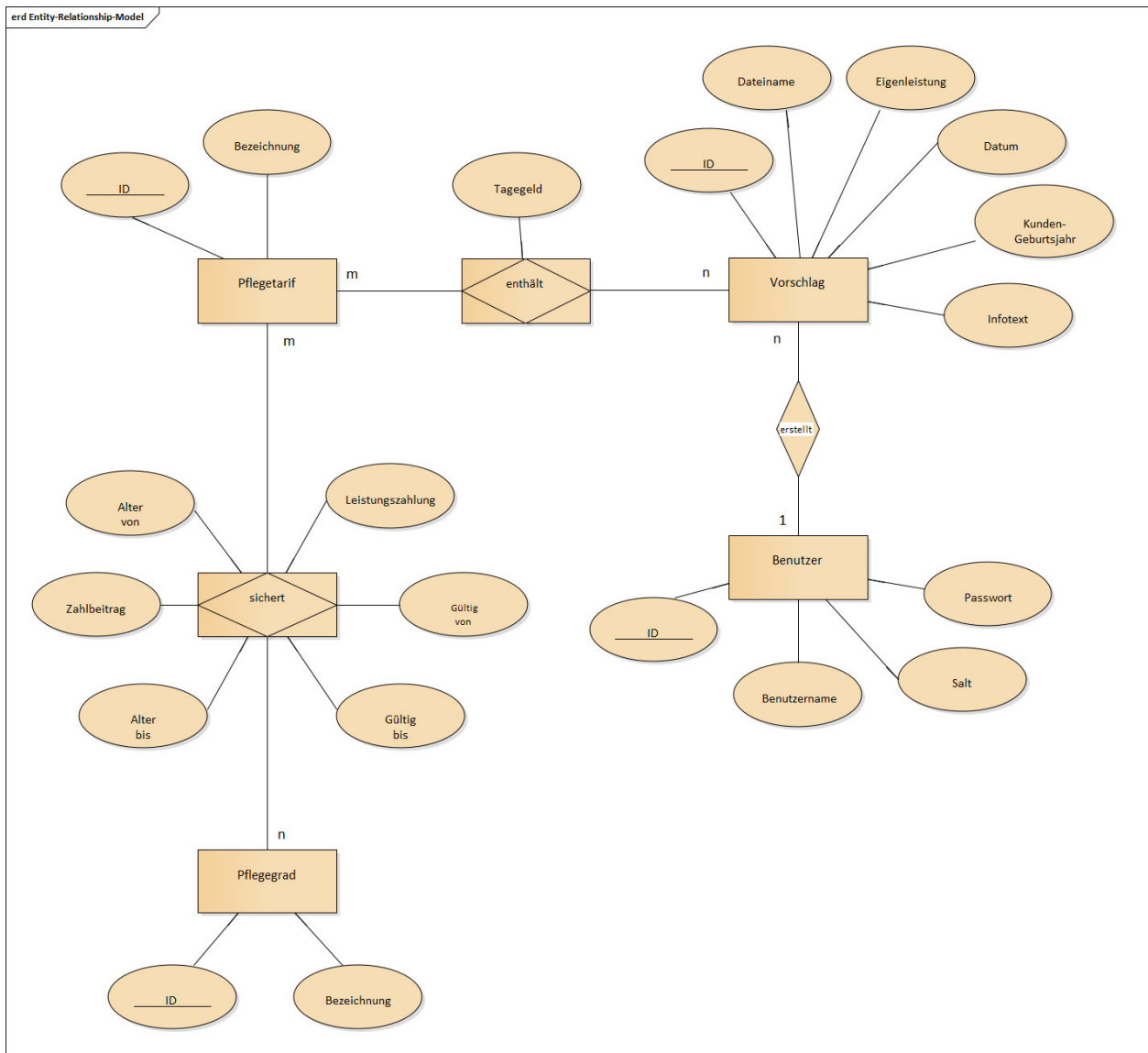
**A.9 Entity-Relationship-Model**

Abbildung 7: Entity-Relationship-Model



## A.10 Tabellenmodell

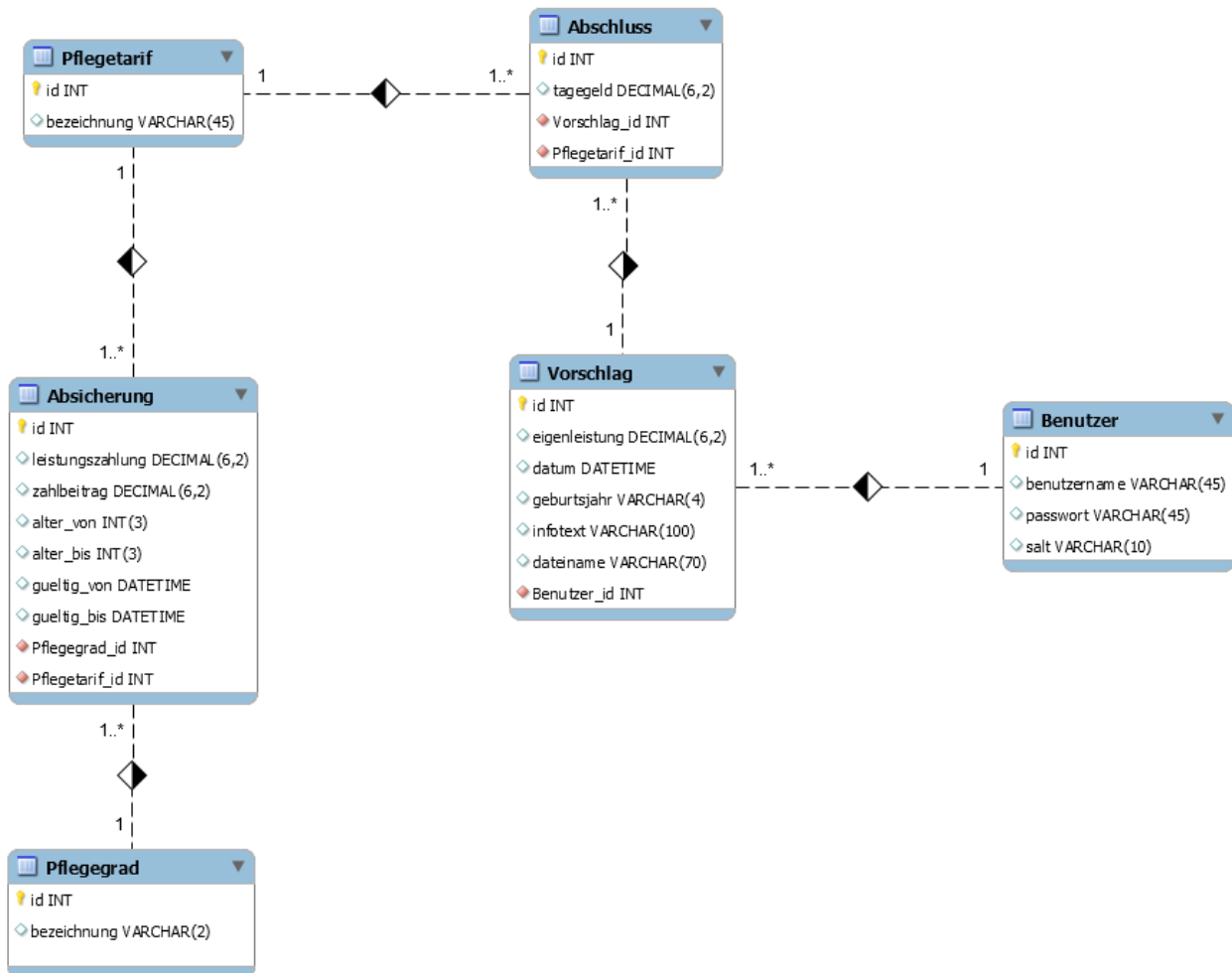


Abbildung 8: Tabellenmodell

## A.11 Aktivitätsdiagramm

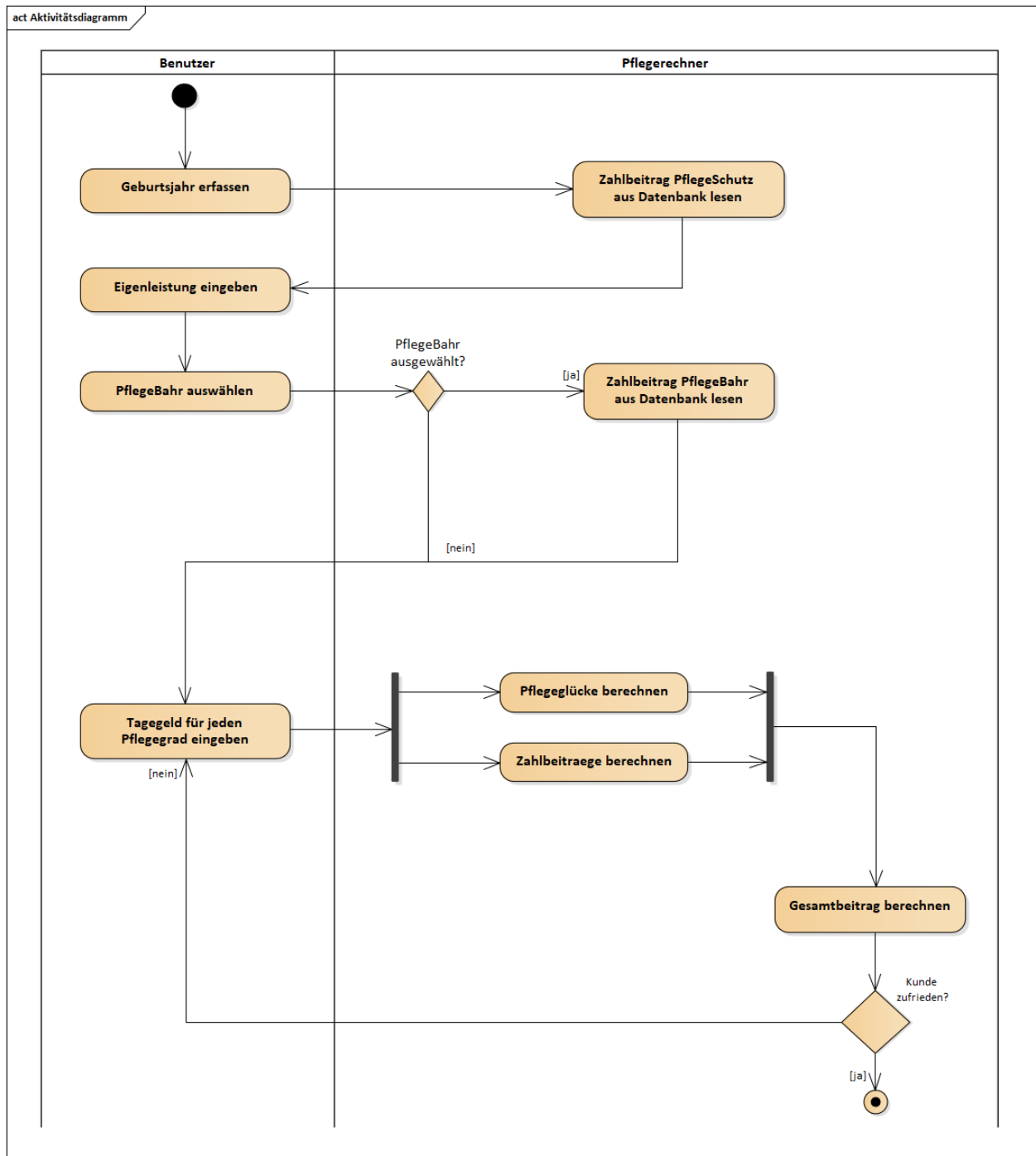


Abbildung 9: Aktivitätsdiagramm des Use-Cases Zahlbeitrag ermitteln

## A.12 Testabdeckung Ansicht aus SonarQube

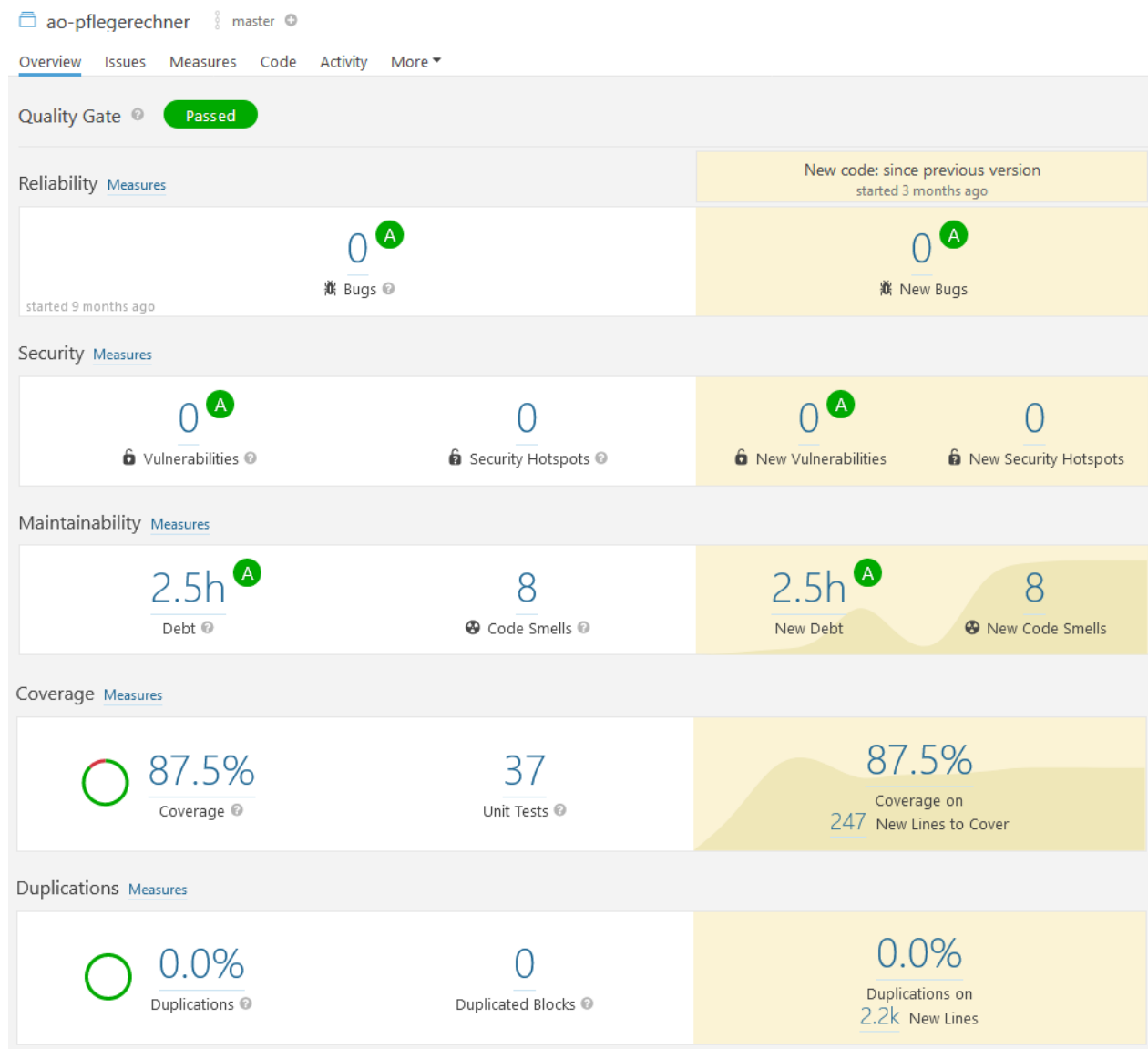


Abbildung 10: Testabdeckung Ansicht aus SonarQube

## A.13 Verteilungsdiagramm

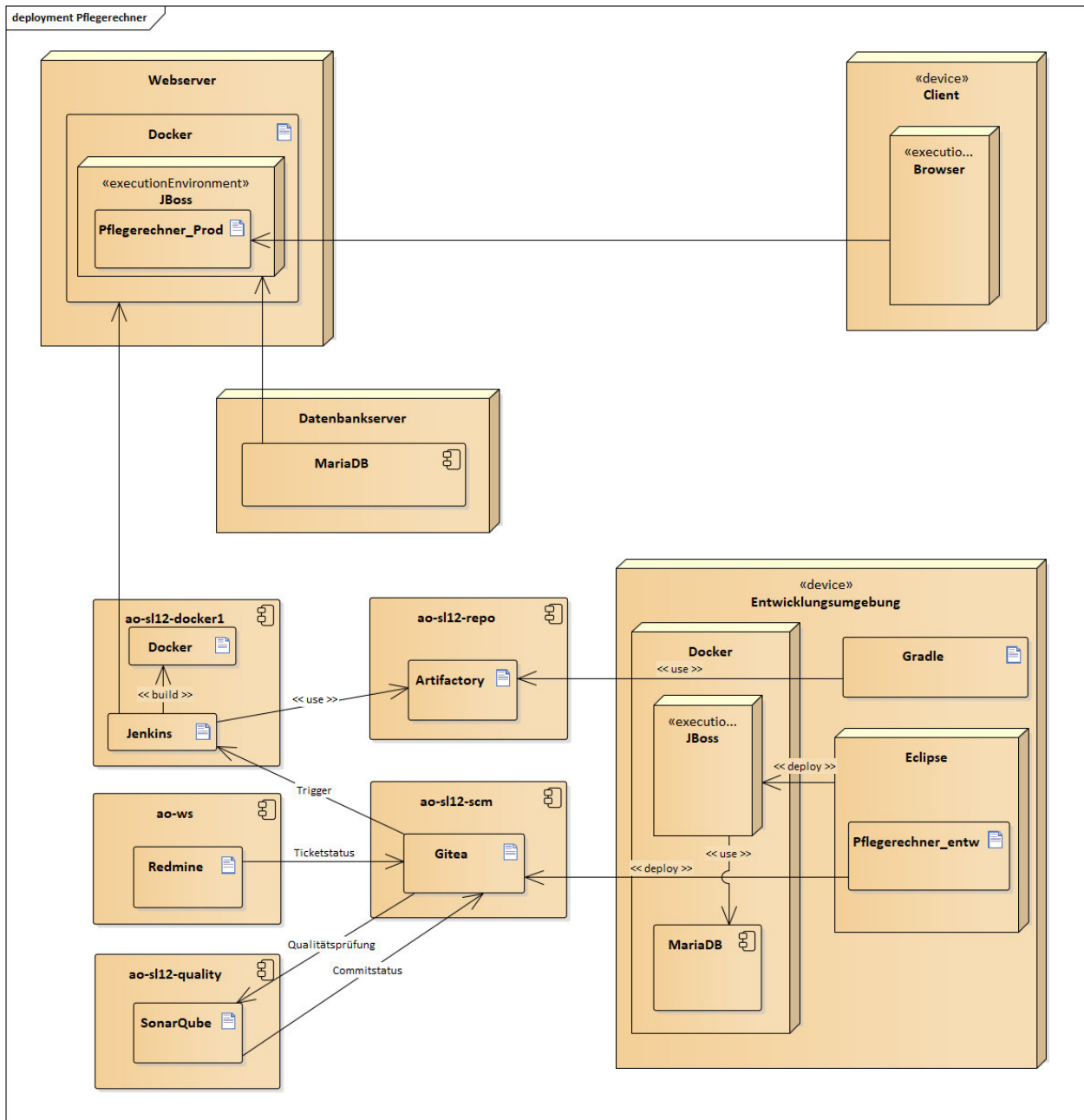


Abbildung 11: Verteilungsdiagramm

**A.14 Jenkinsfile (Auszug)**

```
1 stage ('Publish Image') {
2   steps {
3     script {
4       unstash 'war'
5       docker.withRegistry('http://artifacts.ao-devnet/docker-local', 'tubuild-git') {
6         def image = docker.build('artifacts.ao-devnet/docker-local/ao/pflegerechner:latest')
7         image.push()
8       }
9     }
10  }
11 }
12
13 stage ('Deployment') {
14   agent {
15     ermittleAgent()
16   }
17   steps {
18     script {
19       docker pull artifacts.ao-devnet/docker-local/ao/pflegerechner:latestao-pflegerechner:latest
20       docker stop ao-pflegerechner
21       docker rm ao-pflegerechner
22       docker run -d --name ao-pflegerechner artifacts.ao-devnet/docker-local/ao/pflegerechner:latest
23     }
24   }
25   when {deploymentErlaubt()}
26 }
27
28 def ermittleAgent() {
29   switch (env.BRANCH_NAME) {
30     case 'master':
31       return 'ao-ws-prod'
32     case 'qs':
33       return 'ao-ws-qs'
34     default:
35       return false
36   }
37 }
38
39 def deploymentErlaubt() {
40   switch (env.BRANCH_NAME) {
41     case 'master':
42     case 'qs':
43       return true
44     default:
45       return false
46   }
47 }
```

Listing 1: Auszug aus dem Jenkinsfile

**A.15 JBoss Konfiguration (Auszug)**

```
1 # MariaDB-Treiber hinzufuegen
2 module add
3   --name=org.mariadb.jdbc
4   --resources=${mariadbJar}
5   --dependencies=javax.api
6 /subsystem=datasources/jdbc-driver=mariadb:add
7 ( driver-name=mariadb,
8   driver-module-name=org.mariadb.jdbc,
9   driver-class-name=org.mariadb.jdbc.Driver)
10
11 # DataSource anlegen
12 data-source add
13   --name=PflegerechnerDS
14   --driver-name=mariadb
15   --driver-class=org.mariadb.jdbc.Driver
16   --connection-url="${dburl}"
17   --jndi-name=java:jboss/jdbc/PflegerechnerDS
18   --user-name="${dbuser}"
19   --password="${dbpassword}"
20
21 run-batch
```

Listing 2: JBoss-Konfigurationsdatei

## A.16 Pflichtenheft

### Zielbestimmung

1. Plattform
  - 1.1. Die Anwendung wird mit Java 11 implementiert.
  - 1.2. Für die Webentwicklung kommt die Jakarta-EE-Spezifikation zum Einsatz.
  - 1.3. Die Anwendung wird im Internet erreichbar sein.
  - 1.4. Die Anwendung wird in Docker in einer Instanz des Application Server JBoss-EAP-7.2 ausgeführt.
  - 1.5. Für das Sourcecode Management (SCM) wird Git benutzt.
  - 1.6. Gitea wird als Oberfläche für Repositories verwendet.
  - 1.7. Gradle wird für den automatischen Build-Prozess verwendet.
  - 1.8. Das Deployment wird automatisiert mit einem Jenkins-Server umgesetzt.
2. Datenbank
  - 2.1. Die Datenbankanzubindung erfolgt mit JPA.
  - 2.2. Die Berechnungsdaten werden aus einer Oracle-Datenbank exportiert.
  - 2.3. Für die Web-Anwendung werden die Berechnungsdaten in eine MariaDB importiert.
  - 2.4. Für den Import wird eine REST-Schnittstelle implementiert.
  - 2.5. Die Repository-Klassen implementieren jeweils ein Interface, das die benötigten Methoden vorgibt.
  - 2.6. Die Tests der Repository-Klassen werden mit JUnit 5 geschrieben.
  - 2.7. Für die Tests wird eine eigene Datenbank in einem Docker-Container hochgefahren und mit Testdaten befüllt.
3. Oberfläche
  - 3.1. Die JSF-Technologie wird für die Oberfläche verwendet.
  - 3.2. Die Funktionalität soll auf einer einzigen Seite dargestellt werden.
  - 3.3. Für die Darstellung des Säulendiagramms wird eine JavaScript-Funktion implementiert.
  - 3.4. Die Gestaltung der Oberfläche erfolgt mit Cascading Style Sheets (CSS) 3.
  - 3.5. Beim Oberflächendesign wird die AO-Vorlage zur Einhaltung der Corporate Identity genutzt.
  - 3.6. Die Kommunikation zwischen Browser und Server wird asynchron mit AJAX realisiert. Die Übertragung der Daten wird allerdings in JSON erfolgen.
  - 3.7. Es wird eine Loginmaske geben, die einen Usernamen und ein Passwort abfragt.
  - 3.8. Die Web-Anwendung muss responsiv sein und auf PCs und Tablets laufen.
4. Geschäftslogik
  - 4.1. Für die Erzeugung von Objekten wird CDI verwendet.
  - 4.2. Die Tests werden mit JUnit 5 geschrieben.
  - 4.3. Um die Funktionalität der Repository-Klassen zu simulieren, wird das Mockito-Framework eingesetzt.
  - 4.4. Die Erzeugung der PDF-Datei wird mit der Java-Bibliothek Apache PDFBox umgesetzt.
  - 4.5. Die PDF-Datei wird auf dem Dateisystem des Webserver abgelegt.

**A.17 Listing von JSF-Code**

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
6   xmlns:p="http://primefaces.org/ui"
7   xmlns:h="http://java.sun.com/jsf/html"
8   xmlns:c="http://java.sun.com/jsp/jstl/core"
9   template="/WEB-INF/vorlagen/seite.xhtml">
10  <ui:define name="inhalt">
11    <div id="formular">
12      <ui:include src="/WEB-INF/includes/formular.xhtml" />
13    </div>
14    <div id="untererTeil">
15      <div id="tabellen">
16        <ui:include src="/WEB-INF/includes/pflegeschutz.xhtml" />
17        <h:form id="bahr_checkbox">
18          <div class="pflegebahr-auswahl">
19            <p:selectBooleanCheckbox class="box"
20              value="#{eingabeController.daten.pflegeBahr}"
21              itemLabel="PflegeBahr auswaehlen">
22              <p:ajax event="change" update="pflegebahrtabelle" />
23            </p:selectBooleanCheckbox>
24          </div>
25        </h:form>
26        <p:outputPanel id="pflegebahrtabelle">
27          <c:if test="#{eingabeController.daten.pflegeBahr}">
28            <ui:include src="/WEB-INF/includes/pflegebahr.xhtml" />
29          </c:if>
30        </p:outputPanel>
31        <ui:include src="/WEB-INF/includes/gesetzliche.xhtml" />
32        <ui:include src="/WEB-INF/includes/kosten.xhtml" />
33      </div>
34      <div class="diagramm_wrapper">
35        <ui:include src="/WEB-INF/includes/diagramm.xhtml" />
36      </div>
37      <ui:include src="/WEB-INF/includes/gesamtuebersicht.xhtml" />
38    </div>
39  </ui:define>
40 </ui:composition>

```

Listing 3: Datei: index.xhtml



**A.18 Listing von JavaScript-Code**

```

1 var anzahlAbsicherungen = 4;
2
3 function registriereListener () {
4   for (var i = 0; i <= anzahlAbsicherungen; i++) {
5     document.getElementById("pflegeschutz_table:"+i+":menu").addEventListener("change", function(){
6       aktualisiereDiagramme();
7     });
8   }
9   function aktualisiereDiagramme() {
10    let daten_ambulant = liesDatenAusHTML('ambulant');
11    let daten_stationaer = liesDatenAusHTML('stationaer');
12    zeichneDiagramm('chart_ambulant', daten_ambulant);
13    zeichneDiagramm('chart_stationaer', daten_stationaer);
14  }
15  function liesDatenAusHTML(art) {
16    let pflegeschutz_absicherungen = ermittleAbsicherung(art, "pflegeschutz_table", "pflegeschutz_menu");
17    let pflegebahr_absicherungen = ermittleAbsicherung(art, "pflegebahr_table", "pflegebahr_menu");
18    let gesetzliche_absicherungen = [];
19
20    for (var j = 0; j <= anzahlAbsicherungen; j++) {
21      gesetzliche_absicherung.push(document.getElementById(art + "_wert_" + j).value);
22    }
23
24    if (art === 'ambulant') {
25      let entlastung = 125;
26    }
27
28    let daten = {
29      entlastungsbeitrag: entlastung,
30      gesetzliche_absicherungen,
31      eigenleistung: document.getElementById("eingabe:eigenleistung").value,
32      pflegeschutz_absicherungen,
33      pflegebahr_absicherungen
34    };
35    return daten;
36  }
37  function ermittleAbsicherung(art, tabelle, id) {
38    let absicherung = [];
39    let anzahlTage = 30;
40
41    for (var j = 0; j <= anzahlAbsicherungen; j++) {
42      let tagegeld = document.getElementById(tabelle + ":" + j + ":" + id);
43      pflegeschutz_absicherungen.push(tagegeld.options[tagegeld.selectedIndex].value * anzahlTage);
44    }
45    return absicherung;
46  }

```

Listing 4: Datei: diagramm.js

## A.19 Screenshots der fertigen Anwendung

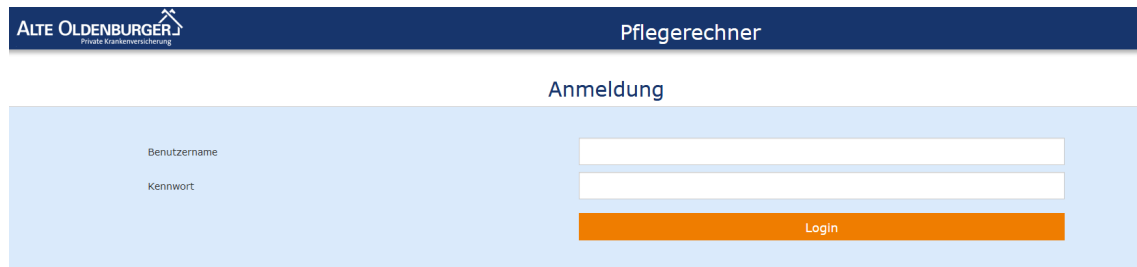


Abbildung 12: Ansicht der Login-Seite

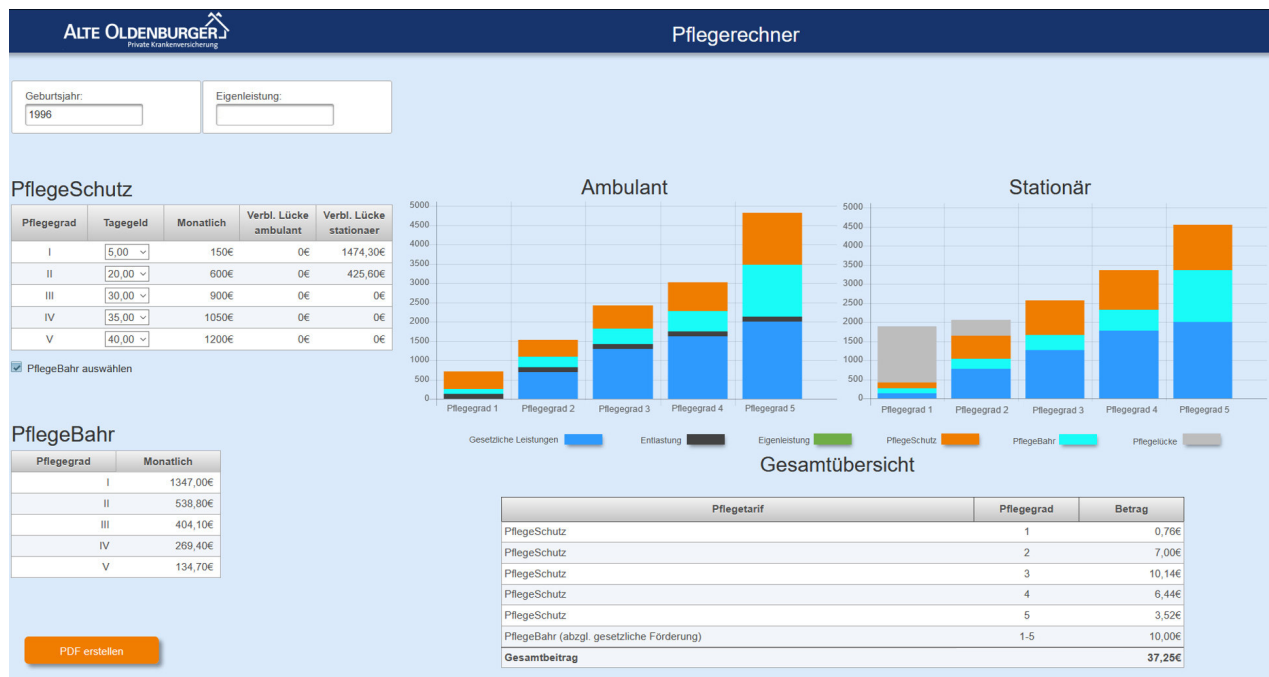


Abbildung 13: Ansicht Anwendung

## A.20 Listing von Java-Code

```

1 @DisplayName("Der PflegelueckeService sollte ")
2 public class PflegelueckeServiceSollte {
3     private PflegelueckeService sut;
4     private AbsicherungRepository absicherungRepository;
5     private Kosten kosten;
6
7     @BeforeEach
8     public void setUp() {
9         absicherungRepository = mock(AbsicherungRepository.class);
10        sut = new PflegelueckeService(absicherungRepository);
11        kosten = new Kosten();
12    }
13
14    private void angenommenDasAbsicherungRepositoryErmitteltDieAbsicherungNachTarifUndGrad(
15        final PflegeSchutzAbsicherung pflegeSchutzAbsicherung, final Absicherung gesetzlicheAbsicherung,
16        final Tarifart tarifart ) {
17        when(absicherungRepository.ermittleAbsicherungNachTarifUndGrad(
18            PflegegradBezeichnung.valueOf(pflegeSchutzAbsicherung.getPflegegrad()), tarifart))
19            .thenReturn(gesetzlicheAbsicherung);
20    }
21
22    @Test
23    @DisplayName("die Pflegeluecke fuer eine Pflegestufe in ambulanter Absicherung ohne PflegeBahr berechnen koennen")
24    public void test01() {
25        final PflegeSchutzAbsicherung pflegeSchutzAbsicherung = new PflegeSchutzAbsicherung();
26        pflegeSchutzAbsicherung.setPflegegrad("III"); //Pflegegrad, in dem die Luecke berechnet werden soll.
27        pflegeSchutzAbsicherung.setTagegeld("15.00"); //Das Tagegeld, das der Kunde abschliesst
28        final BigDecimal gesetzlicheLeistung = BigDecimal.valueOf(1298.00);
29        final boolean pflegeBahrAusgewaehlt = false;
30        final int absicherungId = 1234;
31        final String betragPflegeLuecke = "246.00"; //Ergibt sich aus den geschaetzten Kosten abzuglich allen
32            Leistungen, der Eigenleistung und Entlastungsbeitrag (2119-1298-125-(15*30) = 246)
33
34        final Absicherung gesetzlicheAbsicherung = erzeugeTestAbsicherung(absicherungId,gesetzlicheLeistung);
35        angenommenDasAbsicherungRepositoryErmitteltDieAbsicherungNachTarifUndGrad(pflegeSchutzAbsicherung,
36            gesetzlicheAbsicherung, Tarifart.GKV_AMBULANT);
37        final Pflegeluecke erwartet = new Pflegeluecke(betragPflegeLuecke);
38
39        final Pflegeluecke ergebnis =
40            sut.ermittleLuecke(pflegeSchutzAbsicherung, Tarifart.GKV_AMBULANT, kosten.getKostenAmbulant(),
41                pflegeBahrAusgewaehlt);
42        assertThat(ergebnis).isEqualTo(erwartet);
43    }
44 }

```

Listing 5: Klasse: PflegelueckeServiceSollte

```

1 public class PflegelueckeService implements Serializable {
2     private static final long serialVersionUID = 1641611383225479410L;
3     private static final int ANZAHL_TAGE = 30;
4     private static final BigDecimal ENTLASTUNGSBEITRAG = BigDecimal.valueOf(125.00);
5     private final AbsicherungRepository absicherungRepository;
6
7     @Inject
8     public PflegelueckeService( final AbsicherungRepository absicherungRepository ) {
9         this.absicherungRepository = absicherungRepository;
10    }
11    public Pflegeluecke ermittleLuecke( final PflegeSchutzAbsicherung pflegeSchutzAbsicherung,
12        final Tarifart tarifart, final List<BigDecimal> kosten, final boolean pflegeBahrAusgewaehlt ) {
13        BigDecimal entlastung = BigDecimal.ZERO;
14        BigDecimal pflegeBahrLeistung = BigDecimal.ZERO;
15
16        final BigDecimal gesetzlicheLeistungAmbulant =
17            ermittleGesetzlicheLeistung(pflegeSchutzAbsicherung, tarifart );
18
19        if ( pflegeBahrAusgewaehlt ) {
20            pflegeBahrLeistung = ermittlePflegeBahrLeistung(pflegeSchutzAbsicherung);
21        }
22        if ( tarifart .isAmbulant() ) {
23            entlastung = ENTLASTUNGSBEITRAG;
24        }
25
26        final BigDecimal pflegeSchutzLeistung =
27            BigDecimal.valueOf((pflegeSchutzAbsicherung.getTagegeld())).multiply(ANZAHL_TAGE);
28
29        return new Pflegeluecke(berechnePflegeluecke(pflegeSchutzAbsicherung, kosten, entlastung, pflegeBahrLeistung,
30            gesetzlicheLeistungAmbulant, pflegeSchutzLeistung));
31    }
32    private String berechnePflegeluecke( final PflegeSchutzAbsicherung pflegeSchutzAbsicherung,
33        final List<BigDecimal> kosten, final BigDecimal entlastung, final BigDecimal pflegeBahrLeistung,
34        final BigDecimal gesetzlicheLeistungAmbulant, final BigDecimal pflegeSchutzLeistung ) {
35        return ermittleKosten(pflegeSchutzAbsicherung, kosten)
36            .subtract(pflegeSchutzLeistung)
37            .subtract(pflegeBahrLeistung)
38            .subtract(entlastung)
39            .subtract(gesetzlicheLeistungAmbulant)
40            .setScale(2).toString();
41    }
42    private BigDecimal ermittleLeistung( final PflegeSchutzAbsicherung pflegeSchutzAbsicherung, final Tarifart
43        tarifart ) {
44        return absicherungRepository
45            .ermittleAbsicherungNachTarifUndGrad(pflegeSchutzAbsicherung.getPflegegrad(), tarifart)
46            .getLeistungszahlung();
47    }

```

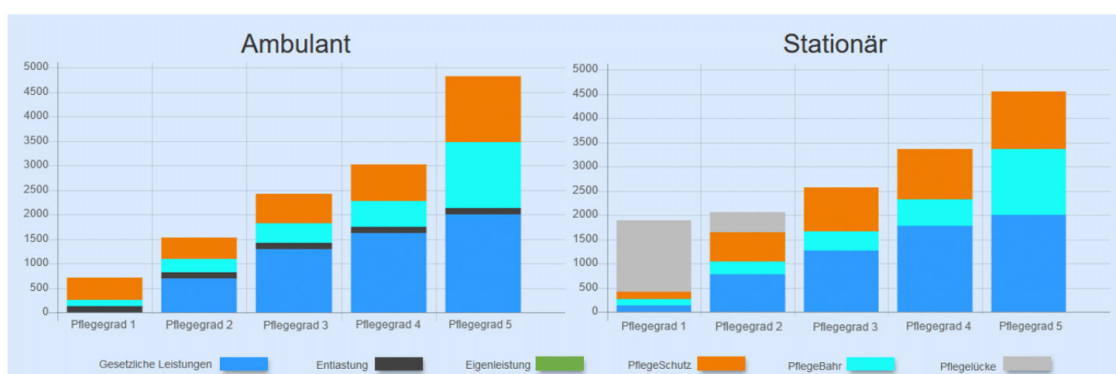
Listing 6: Klasse: PflegelueckeService

## A.21 Benutzerdokumentation (Auszug)

Um die Beiträge zu ermitteln und die Pflegelücke zu berechnen, muss das Geburtsjahr des Kunden im Format „jjjj“ in das Textfeld eingegeben werden. Die Eigenleistung ist der Betrag pro Monat, den der Kunde im Pflegefall selbst aufbringen möchte. Dieser ist optional und kann leer bleiben.



The screenshot shows the top header of the application with the logo 'ALTE OLDENBURGER Private Krankenversicherung'. Below the header, there are two input fields: 'Geburtsjahr:' and 'Eigenleistung:'. Each field has a text input box below it.



Die Diagramme zeigen die Zusammensetzung der einzelnen Leistungen und die verbleibende Pflegelücke. Sobald sich ein Parameter ändert, werden die Diagramme aktualisiert und zeigen die neue Werte an. Die Werte werden detailliert angezeigt, wenn sich der Mauszeiger auf einem Abschnitt des Säulendiagramms befindet.

In der Tabelle PflegeSchutz kann das Tagegeld für den Tarif PflegeSchutz festgelegt werden. Für jeden Pflegegrad kann ein individueller Betrag festgelegt werden. Das Tagegeld in einem Grad muss dabei kleiner oder gleich dem Betrag in dem darüber liegenden Grad sein. Beispielsweise kann das Tagegeld in Pflegegrad IV nicht höher als das festgelegte Tagegeld in Pflegegrad V sein. Das Tagegeld wird mittels Dropdownmenü ausgewählt und ist in 5€-Schritte gestaffelt.

### PflegeSchutz

Pflegegrad	Tagegeld	Monatlich	Verbl. Lücke ambulant	Verbl. Lücke stationaer
I	5,00 ▾	150€	0€	1474,30€
II	20,00 ▾	600€	0€	425,60€
III	30,00 ▾	900€	0€	0€
IV	35,00 ▾	1050€	0€	0€
V	40,00 ▾	1200€	0€	0€

Abbildung 14: Auszug aus der Benutzerdokumentation

## A.22 Entwicklerdokumentation (Auszug)

### Constructor Detail

#### AbsicherungService

```
@Inject  
public AbsicherungService(net.aokv.pflegerechner.model.repository.AbsicherungRepository absicherungRepository)
```

### Method Detail

#### ermittlePflegeAbsicherung

```
public <T extends net.aokv.pflegerechner.view.dto.PflegeAbsicherung> java.util.List<T>  
ermittlePflegeAbsicherung(net.aokv.pflegerechner.model.domain.Tarifart tarifart,  
java.lang.String geburtsjahr,  
java.lang.Class<T> clazz)  
throws net.aokv.pflegerechner.model.exceptions.AbsicherungKonnteNichtErmitteltWerden
```

Ermittelt die Absicherungen der Pflegeversicherung und konvertiert diese in ein Transferobjekt fuer die Darstellung in der Oberflaeche.

**Type Parameters:**

T - Der Typ der Absicherung.

**Parameters:**

tarifart - Die Art des Tarifs.

geburtsjahr - Das Geburtsjahr des Kunden.

clazz - Typ, zu dem die ermittelten Absicherungen gecastet werden.

**Returns:**

Eine Liste von Absicherungen. Nur Subklassen, die von PflegeAbsicherung erben, sind zugelassen.

**Throws:**

net.aokv.pflegerechner.model.exceptions.AbsicherungKonnteNichtErmitteltWerden

#### berechneMonatlicheLeistung

```
public java.lang.Integer berechneMonatlicheLeistung(java.lang.String tagegeld)
```

Berechnet die monatliche Leistung, die der Kunde erhalten wird.

**Parameters:**

tagegeld - Der Geldbetrag, den der Kunde an einem Tag erhalten wird.

**Returns:**

Den Betrag, umgerechnet auf einen Monat.

Abbildung 15: Auszug aus der Entwicklerdokumentation

### A.23 Klassendiagramm der Domäne (Auszug)

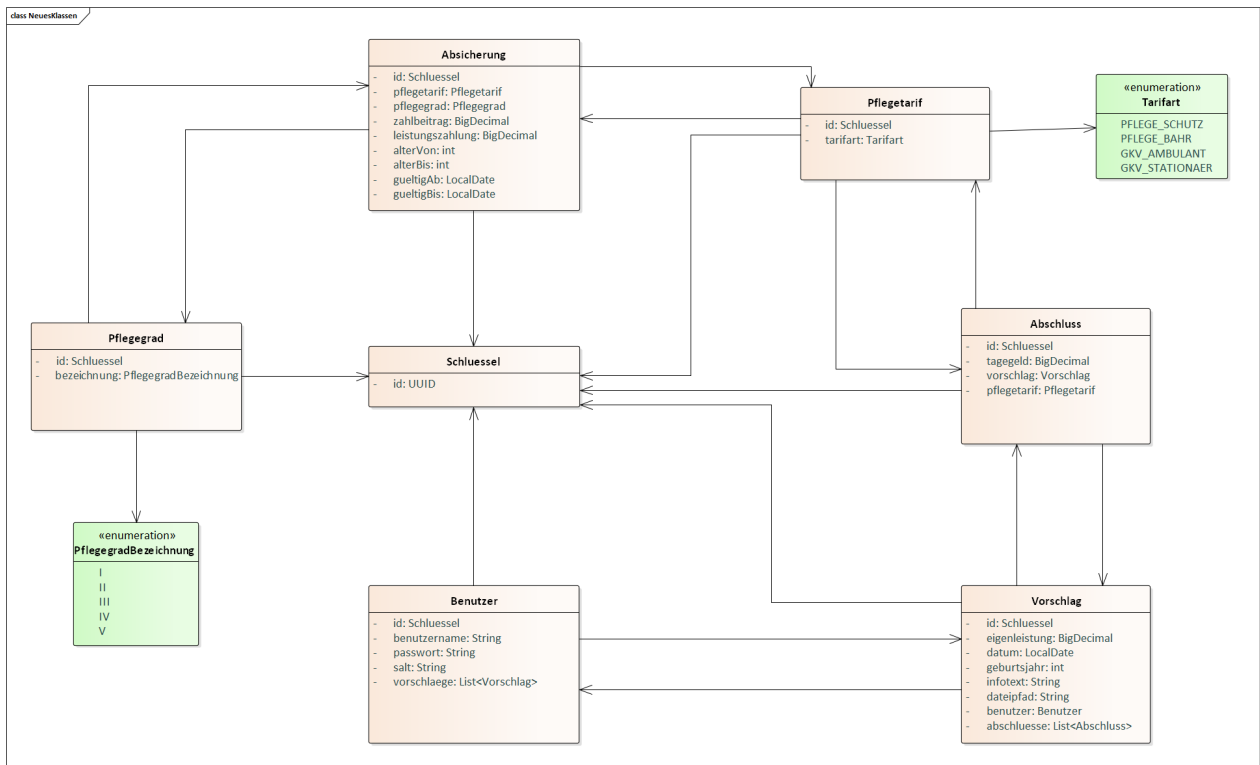


Abbildung 16: Auszug des Klassendiagramms der Domäne