**IPA 2024  
ABSCHLUSSARBEIT**

**OVWEB**

**Offerten-Vergleich**

**17.04.2024**

**Individuelle Praxisarbeit Kanton Zürich**

**Andrei Mititelu**

**Generali AG**

**TBZ**

**IT Apprentices**

**Aschvin Umachandran**

**Sukhnaaz Singh**

**Roger Sorg**

**Roman Babenko**

**Applikationsentwicklung**

**IPERKA**

**Kandidat / Author:**

**Firma:**

**Berufsfachschule:**

**Abteilung:**

**Hauptexperte:**

**Nebenexperte:**

**Berfusbildner:**

**Verantwortliche Fachkraft:**

**Fachrichtung:**

**Projektvorgehensmodell:**

A red circle with a white letter i in it

Description automatically generated

[Teil 1 – Administrativer Teil 4](#_Toc164250811)

[1. Projektaufbauorganisation 4](#_Toc164250812)

[2. Organisation der Arbeitsergebnisse 5](#_Toc164250813)

[2.1 Codesicherung 5](#_Toc164250814)

[2.2 Dokumentationssicherung 9](#_Toc164250815)

[3. Standards 12](#_Toc164250816)

[4. Zeitplan 15](#_Toc164250817)

[5. Arbeitsjournal 16](#_Toc164250818)

[5.1 Tag 1: Mittwoch, 03.04.2024 16](#_Toc164250819)

[5.2 Tag 2: Donnerstag, 04.04.2024 17](#_Toc164250820)

[5.3 Tag 3: Freitag, 05.04.2024 18](#_Toc164250821)

[5.4 Tag 4: Montag, 08.04.2024 19](#_Toc164250822)

[5.5 Tag 5: Dienstag, 09.04.2024 – Halbtag 20](#_Toc164250823)

[5.6 Tag 6: Mittwoch, 10.04.2024 21](#_Toc164250824)

[5.7 Tag 7: Donnerstag, 11.04.2024 22](#_Toc164250825)

[5.8 Tag 8: Freitag, 12.04.2024 23](#_Toc164250826)

[5.9 Tag 9: Montag, 15.04.2024 24](#_Toc164250827)

[5.10 Tag 10: Dienstag, 16.04.2024 – Halbtag 25](#_Toc164250828)

[5.11 Tag 11: Mittwoch, 17.04.2024 26](#_Toc164250829)

[Teil 2 Individueller praktischer Teil 27](#_Toc164250830)

[6. Kurzfassung 27](#_Toc164250831)

[6.1 Ausgangssituation 27](#_Toc164250832)

[6.2 Projektvorgehen 27](#_Toc164250833)

[6.3 Ergebnis 27](#_Toc164250834)

[7. Projektvorgehen 28](#_Toc164250835)

[8. Informieren 29](#_Toc164250836)

[8.1 Ziele der Aufgabenstellung 29](#_Toc164250837)

[8.2 Vorgaben und Anforderungen 29](#_Toc164250838)

[8.3 Ist-Zustand 29](#_Toc164250839)

[8.4 User Stories und Akzeptanzkriterien 46](#_Toc164250840)

[9. Planen 50](#_Toc164250841)

[9.1 Soll-Zustand 50](#_Toc164250842)

[9.2 Mockups 55](#_Toc164250843)

[9.3 Testkonzept 59](#_Toc164250844)

[9.4 Kriterien für die Entscheidungsphase 65](#_Toc164250845)

[10. Entscheiden 66](#_Toc164250846)

[11. Realisieren 67](#_Toc164250847)

[11.1 Datenbank 67](#_Toc164250848)

[11.2 Backend 68](#_Toc164250849)

[11.3 Frontend 79](#_Toc164250850)

[11.4 Automatisiertes Testing 87](#_Toc164250851)

[12. Kontrollieren 90](#_Toc164250852)

[12.1 Manuelles Testing 90](#_Toc164250853)

[12.2 Unit und Jest Tests 94](#_Toc164250854)

[13. Auswerten 96](#_Toc164250855)

[13.1 Vergleich Ist/Soll 96](#_Toc164250856)

[13.2 Persönliches Fazit 96](#_Toc164250857)

[13.3 Schlussreflexion reevaluate 97](#_Toc164250858)

[14. Glossar ALPHABETISCH GEORDNET 98](#_Toc164250859)

[15. Quellenvezeichnis 99](#_Toc164250860)

[15.1 Einsatz von generativen Tools 100](#_Toc164250861)

[16. Danksagung 101](#_Toc164250862)

Teil 1 – Administrativer Teil

## Projektaufbauorganisation

**Lehrbetrieb und Durchführungsort**Generali AG   
Soodmattenstrasse 4   
8134, Adliswil   
[058 472 75 00](https://www.google.com/search?client=firefox-b-d&q=generali+adliswil+&lqi=ChFnZW5lcmFsaSBhZGxpc3dpbCIDiAEBSKmIiN3lgICACFofEAAYABgBIhFnZW5lcmFsaSBhZGxpc3dpbCoECAIQAJIBEGluc3VyYW5jZV9hZ2VuY3mqAVcKDS9nLzExYjY2anJoc2QQASoMIghnZW5lcmFsaSgFMh8QASIb7rjLivujVg1VYAHWgXdZ_l1vC9h1_he9K8OnMhUQAiIRZ2VuZXJhbGkgYWRsaXN3aWw)  
[life.ch@generali.com](mailto:life.ch@generali.com)  
  
  
**Hauptexperte**Aschvin Umachandran   
aschvin00@gmail.com   
078 826 61 57  
  
  
**Verantwortliche Fachkraft:**Roman Babenko  
[Generali](mailto:marc.siegmund+pkorg@gmail.com) AG  
Soodmattenstrasse 4  
8134, Adliswil  
076 410 87 29  
roman.babenko@generali.com  
  
  
**Auftraggeber:**Daniel Sager  
Generali AG  
Soodmattenstrasse 4  
8134, Adliswil  
daniel.sager@generali.com

**Kandidat**Andrei Mititelu   
Breitwiesstrasse 15   
8135, Langnau am Albis   
079 343 49 40  
[andrei@gimli.ro](mailto:andrei@gimli.ro)  
  
  
**Nebenexpertin**Sukhnaaz Singh  
sukhnaazsingh@hotmail.com  
076 411 42 35  
  
  
**Berufsbildner / Lehrfirma:**Sorg Roger  
[Generali](mailto:marc.siegmund+pkorg@gmail.com) AG  
Soodmattenstrasse 4  
8134, Adliswil  
058 472 40 42  
roger.sorg@generali.com

## Organisation der Arbeitsergebnisse

Zur Vermeidung von Datenverlusten im Falle eines Systemausfalls wird während der IPA das Versionierungssystem GitHub verwendet. Dieses System ermöglicht auch den Zugriff auf frühere Versionen der Dokumente und Dateien.

### Codesicherung

Für den Quellcode wird ein eigenständiger Branch innerhalb des Monorepository des Projekts angelegt. Sämtliche Änderungen am Code oder an der Projektdokumentation werden auf diesem Branch hochgeladen und dort gesichert.

**Link zum Branch:** <https://github.com/IT-Apprentices/insurance-offer-management/tree/OVB-170-Offerten-Vergleich-IPA>

Nachfolgend ist die Historie des Branches dargestellt:

commit afea698c2feb9572c1eae1cc052e72761b4b1451 (HEAD -> OVB-170-Offerten-Vergleich-IPA, origin/OVB-170-Offerten-Vergleich-IPA)

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 15 11:10:10 2024 +0200

feat(be): Fixing up tests

commit 409905bcace872fb3b561221fe1c6d21ccb3dc9f (HEAD -> OVB-170-Offerten-Vergleich-IPA, origin/OVB-170-Offerten-Vergleich-IPA)

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 16:15:56 2024 +0200

feat(be): implementing backend unittesting for addCustomerInfoToRequest and checkOfferValidity

commit 5c931d5f77a21d236decfa5171002c03d3859427

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 16:14:41 2024 +0200

feat(fe): frontend tests and storybook implementing

commit ae772daf6a903472fba4161bbba673e5269f2f77

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 12:56:52 2024 +0200

feat(fe): implementing front end tests

commit 45094cdb830086d4d1723cef9cee7e4db0011b52

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 09:57:13 2024 +0200

feat(fe): Added tracing and logging to frontend actions

commit 57d37a2f17f8e174747b7daa866262cc44b22d2d

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 09:56:58 2024 +0200

feat(be): Added tracing and logging to backend functions

commit 7b6b7f5ffcab09819d45a19104f0c6d52d01ef44

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 08:21:28 2024 +0200

feat(be): Implemented logging into new methods and into the old ones as was seen fit

commit d5919aad6e65b8ba57f299af0d6e1b35399c1ed4

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 10 15:58:48 2024 +0200

feat(be): Small changes using ESLint, implementing of errorhandling in create-insurance-form.tsx using form setError

commit 4d802d2cc35ab22e58d74cfd00ae0bce37491640

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 10 14:16:39 2024 +0200

feat(be): Removing NotNull Annotation from fields where it shouldn't be there

commit 8c9983d8fe1d8e200806fd3e58ad7af6bcb929a1

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 10 13:50:32 2024 +0200

feat(be): Refactoring, removed an unnecessarily added model, reformated to use maptofosoftrequest in validateOffer instead of having its own mapper, which was redundant

commit 40b1e96e512159092019c8d6a90a6784b9579ea7

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 10 12:14:55 2024 +0200

feat(be): Adjusting component to look as designed, implementing validation on both component and parent page

commit f126f878110d54b9c6ef9cb05899799c4f88d7ab

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 10 12:14:14 2024 +0200

feat(be): Mapper,tests and dto being adjusted to accomodate possibility of proper validation in frontend on detail page

commit c6292b17cf3720816585b41c808f6a4d936ae7eb

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Tue Apr 9 17:48:10 2024 +0200

feat(fe): Creating a new component for the calculated offers, utilizing this component in offer details page and removing the old outdated code.

commit 9b5927c3fcbbd0a7f626bb7a51065fd109a17f51

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Tue Apr 9 14:56:25 2024 +0200

feat(fe): Fixing validation and small ESLint code mistakes. Form now working accordingly.

commit 584033a37c229c7c9f9691a5a38a98e3e5628679

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Tue Apr 9 14:55:48 2024 +0200

feat(be): Implementing Error Handling for validating a life insurance request. Backend should return error from FOSOFT API to frontend without crashing.

commit c11dbc624b2e2f418eb213c75557df45d0d4455e

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 16:12:53 2024 +0200

feat(be): Adjusting backend to be able to calculate up to 3 offers

commit aa7dbaa65972575cc83a3f5ae42376fea433fe9c

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 14:23:21 2024 +0200

feat(be): Adjusting frontend to have 3 values for versicherungssumme, also adjusting to handle error with validation, changes only in create-insurance-form.tsx

commit 964cdc19190d607d431dfb150fa02e37afd70eb0

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 14:22:35 2024 +0200

feat(be): Adjusting controllers and services to handle more than 1 value upon creation of a new offer. Also adjusting them to return an error when business rules are violated

commit 0e5c37fce780de9dc8e8d5927a76b2cfeef46d16

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 14:21:19 2024 +0200

feat(be): Updating mappers and dtos, adjusting test to fit them

commit d810a761e628dbeea98c78f1fa252a9d608ce56d

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 09:20:05 2024 +0200

feat(be): altering Models, DTOs and Mappers to be able to handle multiple VS values. Creating a new model for posting a life insurance for validation.

commit 4374ed6f366b437871661e87b63eebb4a5a7426f

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 09:18:16 2024 +0200

feat(be): include versicherungssumme (vs) 1, 2 and 3 on life\_insurance\_request table

### Dokumentationssicherung

Für die Sicherung aller Dateien für die Dokumentation wird ein separates GitHub Repository erstellt. Auf diesem wird täglich gepusht. Ebenfalls werden die Dokumentation und der Zeitplan täglich lokal abgespeichert, um somit auch lokal eine Versionierung zu haben.

Auf Anfrage des Hauptexperten wurde die Dokumentation auch jeden zweiten Tag als PDF auf PkOrg hochgeladen.

**Link zum Repository:** <https://github.com/Rad14nt/IPA_2024_Docs>

Nachfolgend ist die Historie des Branches dargestellt:

commit 9a48201fdd57c8c49c5f71553af7ea46e6b26137 (HEAD -> main, origin/main)

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Tue Apr 16 17:43:54 2024 +0200

Tag 10 - Backup Commit und Push

commit 24bf218fffd3787fd7eaa2c7d6d32e46cd070cfe

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 15 16:41:06 2024 +0200

Tag 9 - Backup Commit und Push

commit 3c12620e30dc3d2ab2295ba23726ee314418c6de (HEAD -> main, origin/main)

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Fri Apr 12 17:14:13 2024 +0200

Tag 8 - Backup Commit und Push

commit 359c21475457042071c5de90789314fa334a1543

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 11 16:43:24 2024 +0200

Tag 7 - Backup Commit und Push

commit bfbac69f1486c88047fefe8cfb2bd40de409965f

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 10 17:02:35 2024 +0200

Tag 6 - Backup Commit und Push

commit c5437abdd08b135299dc3d9e51af10a30d4cc828

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Tue Apr 9 18:14:50 2024 +0200

Tag 5 - Backup Commit und Push

commit 97e547645db607d3fc6c4014f678e80091cf592e

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Mon Apr 8 16:51:39 2024 +0200

Tag 4 - Backup Commit und Push

commit f912031f80f9ce183eb3b552ccd4f33db3ce59fc

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Fri Apr 5 17:27:20 2024 +0200

Tag 3 - Backup Commit und Push

commit f7e328492fcef6da381c6c94ecfac7cfe1739bc6

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Thu Apr 4 16:25:34 2024 +0200

Tag 2 - Backup Commit und Push

commit 55561ba205d5f6cb0ccbf66c0995e15558293855

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 3 17:10:47 2024 +0200

Tag 1 - Backup Commit und Push

commit 487d15b3ac254d5055a3c97a0b58bb91302e61da

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 3 17:10:36 2024 +0200

Tag 1 - Backup Commit und Push

commit cd3ac4117dbf5ff41b603f18d407428125c3a2bb

Author: Andrei Mititelu <andrei@gimli.ro>

Date: Wed Apr 3 12:39:55 2024 +0200

Zeitplan fertig und hochgeladen

Folgend der Beweis für die lokale Speicherung:

A screenshot of a computer

Description automatically generated

## Standards

Im Rahmen des Projektes OVWEB und dieser IPA werden folgende Technologien eingesetzt:

|  |  |
| --- | --- |
| **Frontend** | |
| Name | Einsatz |
| React 18 | Frontend UI Framework |
| NextJS 14 | React-based Framework |
| Typescript | Frontendsprache |
| Jest | Testing Framework |
| Storybook | Development und Testing Framework |
| Shadcn/ui | Komponenten Library |
| Prettier und ESLint | Code Formattierungs- und Analysetool |
| NextAuth | Authentifizierung |

|  |  |
| --- | --- |
| **Backend** | |
| Name | Einsatz |
| Java | Backendsprache |
| Springboot | Java-Framework |
| Lombok | Annotationen |
| JUnit | Testing |
| Hibernate | ORM-Framework |

|  |  |
| --- | --- |
| **Daten Layer** | |
| Name | Einsatz |
| PostgreSQL | Datenbank |
| Flyway | Datenbank-Versionierungstool |

|  |  |
| --- | --- |
| **Identity Provider** | |
| Name | Einsatz |
| OpenID Connect | Identitätsprotokoll |
| Keycloak | Identitäts- und Zugriffsmanagement |

|  |  |
| --- | --- |
| **Entwicklungsumgebung** | |
| Name | Einsatz |
| JetBrains IntelliJ IDEA | Entwicklungsumgebung |
| Docker | Containerisierungsplatform |
| Windows 11 | Operating System |

|  |  |
| --- | --- |
| **UI/UX Design** | |
| Name | Einsatz |
| Figma | Design Tool |
| Generali-Styleguide | Styleguide |

|  |  |
| --- | --- |
| **Dokumentation und Administration** | |
| Name | Einsatz |
| Microsoft Office (Word und Excel) | Dokumentation |
| Draw.io und IntelliJ IDEA | Diagramme bzw. Skizzen |
| OneDrive | Automatische Speicherung |
| Github | Versionierung der Dokumentation und Backups |
| ChatGPT | Textgenerierung |
| Duden.de | Rechtschreibprüfung |
| LanguageTool.org | Rechtschreibprüfung |

|  |  |
| --- | --- |
| **Projektmanagement und Development Tools** | |
| Name | Einsatz |
| Jira | Projektmanagement-Tool |
| Github | Versionskontroll- und Kollaborationsplattform |

## Zeitplan

## Arbeitsjournal

In der Sektion „Heutige Arbeiten“ werden die Aufgaben aufgelistet, an denen heute gearbeitet wurde. Es wird unterschieden zwischen abgeschlossenen und noch laufenden Aufgaben:

✔ **Abgeschlossen**: Diese Aufgaben wurden heute vollständig abgeschlossen. Es sind keine weiteren Arbeiten an dieser Aufgabe erforderlich.

➜ **In** **Bearbeitung**: An diesen Aufgaben wurde gearbeitet, sind jedoch noch nicht abgeschlossen. Die Arbeit an diesen Aufgaben wird fortgesetzt.

### Tag 1: Mittwoch, 03.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Aufgabenstellung und Anforderung studieren * Zeitplan erstellen * Ist-Zustand Analyse * Dokumentation – Teil 1, Administrativer Teil * Dokumentation – Informieren * Dokumentation – Arbeitsjournal | | * Aufgabenstellung und Anforderung studieren * Zeitplan erstellen * Ist-Zustand Analyse * Dokumentation – Teil 1, Administrativer Teil * Dokumentation – Informieren * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Für den Zeitplan lag keine Vorlage vor, deshalb nahm diese Aufgabe mehr Zeit in Anspruch als geplant. | |
| **Erfolge** | Der Start in die IPA verlief dank der vorhandenen Dokumentationsvorlage zügig, und die sorgfältige Erstellung des Zeitplans ermöglichte mir einen strukturierten Einstieg in die IPA. | |
| **Misserfolge / Probleme und Lösungen** | Beim Aufsetzen meines Projektes stiess ich auf Probleme mit einer neuen Implementierung von SendGrid. Es fehlten die Login-Daten in der Umgebungsvariablen-Datei und ich konnte somit das Backend nicht starten, um Screenshots für meine IST-Analyse zu machen. Ich konnte das Problem lösen, indem ich meinen Kollegen Samuel bat, mir diese Daten aus seinem Projekt zu kopieren, da diese Daten nicht in der Dokumentation des Projektes zu finden waren. | |
| **Hilfestellung und Begründung** | Da die Login-Daten für SendGrid in der Dokumentation fehlten, musste ich meinen Kollegen Samuel bitten, mir diese aus seinem Projekt zu übermitteln, da er SendGrid implementierte und die Login-Daten für alle gleich sind. | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Trotz anfänglichem Stress aufgrund bestehender Unsicherheiten bin ich mit dem Fortschritt des heutigen Tages zufrieden und freue mich, dass meine IPA weitgehend nach Plan verläuft. Nach diesem Tag konnte ich mich ein wenig beruhigen, da ich einen Plan habe und dieser auch sehr sorgfältig erstellt wurde, was mir Sicherheit gibt. | |

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * User Stories und Akzeptanzkriterien * Dokumentation – Informieren * Soll-Zustand definieren inkl. Diagramme * Mockups erstellen (Figma) * Dokumentation – Planen * Dokumenation – Arbeitsjournal | | * User Stories und Akzeptanzkriterien * Dokumentation – Informieren * Soll-Zustand definieren inkl. Diagramme * Mockups erstellen (Figma) * Dokumentation – Planen * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Das Dokumentieren der Informationsphase dauerte etwa eine halbe Stunde länger als erwartet. Dies lag daran, dass ich eine bestehende Funktion erweitern muss. Daher musste ich mehr dokumentieren, als ich anfangs gedacht hatte. | |
| **Erfolge** | Ich konnte den Soll-Zustand schneller als geplant fertigstellen, da ich mir im Voraus Gedanken zur IPA gemacht hatte.  Ich konnte auch meine Mockups rechtzeitig zeichnen sowie fast vollständig dokumentieren. | |
| **Misserfolge / Probleme und Lösungen** |  | |
| **Hilfestellung und Begründung** |  | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der heutige Tag verlief fast genau nach Plan, was mir ein hohes Mass an Zufriedenheit und ein Gefühl der Erleichterung brachte.  Ich hatte mir die Nacht zuvor Sorgen gemacht, ob meine Zeiteinschätzung richtig war. Ich konnte qualitativ hochwertige Arbeit leisten, insbesondere bei der Erstellung der Mockups  Dies war auch wichtig, da die Benutzerfreundlichkeit bei dem Vergleich im Vordergrund der IPA steht.  Diese Effizienz ermöglichte es mir, ohne grossen Druck und mit einem klaren Kopf zu arbeiten, was zu einer insgesamt angenehmen und produktiven Arbeitsatmosphäre führte. Die Tatsache, dass ich keine Unterstützung benötigte und keine wesentlichen Hindernisse auftraten, verstärkte mein Vertrauen in meine Fähigkeiten und die Planung des Projekts.  Für morgen bin ich gut aufgestellt und sehe den weiteren Aufgaben mit Optimismus entgegen. | |

### Tag 2: Donnerstag, 04.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Dokumentation – Planen * Model der Datenbank erstellen * Testkonzept erstellen * Expertenbesuche * Lösungsvariante festlegen * Dokumentation – Entscheiden * Dokumentation – Arbeitsjournal | | * Model der Datenbank erstellen * Testkonzept erstellen * Lösungsvariante festlegen * Dokumentation – Planen * Dokumentation – Entscheiden * Expertenbesuche * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Die korrektur der fehlerhaften Dokumentation der ANGEBOTsoftware in Kapitel 8.3.3 und die Anpassung der Schriftarten in verschiedenen Kapiteln der Dokumentation.  Es gab etwas mehr Zeitaufwand bei der Dokumentation im Planungsteil, jedoch konnte ich durch die schnellere Fertigstellung der Lösungsvariante für die Detailseite Zeit einsparen. | |
| **Erfolge** | Schnelle Erstellung des Datenbankmodells und zügige Formulierung des Testkonzepts aufgrund klar definierter Akzeptanzkriterien.  Mein erstes Expertengespräch fand statt, und die freundliche Art der Experten half mir, meine Nervosität zu überwinden. | |
| **Misserfolge / Probleme und Lösungen** | Die Antwort der ANGEBOTsoftware in Kapitel 8.3.3 war fehlerhaft dokumentiert und benötigte Korrekturen. Ebenso musste die Schriftart in einigen Kapiteln der Dokumentation berichtigt werden. | |
| **Hilfestellung und Begründung** |  | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der Tag verlief insgesamt entspannt und produktiv. Trotz anfänglicher Herausforderungen, wie der Korrektur von Dokumentationsfehlern, konnte ich effektiv arbeiten und wesentliche Fortschritte erzielen. Mein erstes Expertengespräch trug wesentlich zur Beruhigung und positiven Stimmung bei, und die freundliche Unterstützung der Experten stärkte mein Selbstvertrauen. Das erfolgreiche Abschliessen der Aufgaben gab mir zusätzliche Zuversicht für die kommenden Herausforderungen. Ich freue mich zuerst auf das Wochenende und die danach folgende Realisierungsphase. | |

### Tag 3: Freitag, 05.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Backend:Datenbank * Backend: Model und DTOs * Backend: Controller und Services * Dokumentation – Realisieren * Dokumentation – Arbeitsjournal | | * Backend: Datenbank * Backend: Model und DTOs * Backend: Controller und Services * Dokumentation – Realisieren * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Heute hatte ich unerwartete Änderungen an der Erstellungsseite vorgenommen, um das Fehlerhandling später gleichzeitig sowohl im Backend als auch im Frontend zu implementieren, was mir an Zeit ersparen wird. | |
| **Erfolge** | Das Migrationsskript wurde erfolgreich ausgeführt. Zudem wurde die Implementierung eines Offerten-Bundles so wie auch dessen Validierung im Backend abgeschlossen und die Anpassungen an der Erstellungsseite wurden erfolgreich durchgeführt. Die Erstellungsseite zeigt auch den Fehler, der vom ANGEBOTSoftware bei der Verletzung der Business-Rules zurückkommt. | |
| **Misserfolge / Probleme und Lösungen** | Ich stiess auf Probleme mit der Ausführung meines Flyway-Skripts, weil das Skript aufgrund eines Fehlers in der Benennung nicht erkannt wurde. Durch die Überprüfung der Dokumentation konnte ich das Problem identifizieren und lösen, es fehlte einen Unterstrich. | |
| **Hilfestellung und Begründung** | Zur Lösung des Problems mit dem Flyway-Skript nutzte ich die Flyway-Dokumentation, wo ich merkte, dass zwei Underscores nötig waren.  Für das Implementieren des Fehlerhandlings verwendete ich zwei Quellen. Eine, die sich mit der Handhabung von ResponseStatusException befasst und eine, die im Frontend den Fehler auf dem Form setzt. Diese Schritte waren notwendig, damit ich den Fehler vom ANGEBOTSoftware im Frontend anzeigen konnte. | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der heutige Tag war herausfordernd, aber lehrreich.  Trotz anfänglicher Rückschläge konnte ich die Probleme effektiv lösen und wichtige Fortschritte in der Entwicklung der Anwendung erzielen.  Ich bin froh, das erste Feature, die Validierung und Erstellung eines Offerten-Bundles, implementiert zu haben. | |

### Tag 4: Montag, 08.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Frontend: Anpassungen an der Create-Offer Seite * Dokumentation – Realisieren * Dokumentation – Arbeitsjournal | | * Backend: Controller und Services * Frontend: Anpassungen an der Create Offer Seite * Frontend: Anpassungen an der Offer-Detail Seite * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Statt wie geplant mit der Dokumentation fortzufahren, fokussierte ich mich auf die Anpassung von Controllern und Services, die während der Implementierung des Frontends doch noch ständige Anpassungen brauchen. Dies war aus dem Grunde, dass die Dokumentation immer wieder wegen der ständigen Änderungen an Controller und Services neugeschrieben werden musste. Um diesen Zeitverlust zu minimieren, werde ich deshalb die Dokumentation für die Realisierungsphase erst am Schluss schreiben. | |
| **Erfolge** | Ich konnte das Fehlerhandling in den Controllern und Services verbessern und erfolgreich eine neue wiederverwendbare Komponente für die Offer-Detailseite entwickeln. Zudem wurde die Erstellungsseite schneller als erwartet fertiggestellt, und ich machte rasche Fortschritte bei der Implementierung der Liste von Offerten auf der Detailseite. | |
| **Misserfolge / Probleme und Lösungen** | Das Design der neuen Komponente entspricht noch nicht exakt dem Figma-Design, was eine weitere Anpassung erfordert. | |
| **Hilfestellung und Begründung** | Zur Gestaltung der neuen Komponente nutzte ich die Dokumentation von TailwindCSS, um eine genaue Übereinstimmung mit meinem Figma-Design zu erreichen. | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der heutige Halbtag war produktiv, trotz des stressigen Starts nach dem Mittagessen. Die Entscheidung, die Dokumentation zu verschieben, erwies sich als effektiv, um kritische Anpassungen vorzunehmen und die Entwicklung voranzutreiben. Dies bestärkt die Wichtigkeit flexibler Prioritätensetzung in Phasen intensiver Entwicklung. | |

### 

### Tag 5: Dienstag, 09.04.2024 – Halbtag

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Frontend: Anpassungen an der Create-Offer Seite * Frontend: Anpassungen an der Offer-Detail Seite * Implementierung Logging * Dokumentation – Realisieren * Dokumentation – Arbeitsjournal | | * Backend: Anpassungen an Controller und Services * Frontend: Anpassungen an der Create-Offer Seite * Frontend: Anpassungen an der Detail Seite * Implementierung Logging * Tests schreiben * Dokumentation – Realisieren * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Aufgrund von Fehlern in den Backend-Tests, die durch die neuen Versicherungssummen verursacht wurden, musste ich unvorhergesehene Korrekturen vornehmen, was eine halbe Stunde beanspruchte. | |
| **Erfolge** | Ich habe erfolgreich Anpassungen am Fehlerhandling auf der Erstellungsseite durchgeführt und das Design der Detailseite eng an das Figma-Mockup angelehnt. Zusätzlich wurden nach der Korrektur der Backend-Tests Anpassungen am Modell vorgenommen, um die Validierung im Frontend korrekt zu unterstützen.  Somit sind die zwei Hauptfunktionen der IPA aus der User Perspektive implementiert. | |
| **Misserfolge / Probleme und Lösungen** | Ich stiess auf Probleme mit den Backend-Tests, die durch Anpassungen an den Versicherungssummen ausgelöst wurden. Nach kurzen Anpassungen war dies jedoch repariert.  Ausserdem war die Dokumentation der durchgeführten Änderungen aufwändiger als erwartet, was mich dazu veranlasste, mehr Zeit dafür einzuplanen. | |
| **Hilfestellung und Begründung** | Zur Gestaltung der neuen Komponente nutzte ich die Dokumentation von TailwindCSS, um eine genaue Übereinstimmung mit meinem Figma-Design zu erreichen.  Ich benutzte ebenfalls die Dokumentation für die Error State auf dem Form, die ich schon im Tag 4 benutzt hatte, um das Fehlerhandling zu verbessern. | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Trotz einiger Herausforderungen war der Tag produktiv.  Die Notwendigkeit, die Tests zu überarbeiten, hat mir gezeigt, dass es schwierig sein kann, alles bei der Planung im Voraus zu sehen.  Vielleicht würde es sich lohnen, in der Zukunft für jede IPERKA Phase eine halbe Stunde als Buffer zu nehmen. In diesem Fall hatte ich Glück, dass ich durch schnelleres Erledigen anderer Aufgaben auf dem Zeitplan bleiben konnte.  Das Abschliessen der Implementation der Detail- und Erstellungsseiten war jedoch eine Erleichterung. Es zeigte mir, dass ich für die Implementierung im Frontend meine Zeit gut eingeschätzt habe. | |

### Tag 6: Mittwoch, 10.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Implementierung Logging * Implementierung Tracing * Tests schreiben * Expertenbesuche * Dokumentation – Arbeitsjournal | | * Implementierung Logging * Implementierung Tracing * Expertenbesuche * Dokumentation – Realiseren * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Der zweite Expertenbesuch dauerte eine halbe Stunde weniger als eingeplant.  Ich arbeitete danach am Logging und Tracing, welche beide schneller fertig waren als geplant. Dies erlaubte es mir, frühzeitig mit den Tests anzufangen und diese schon heute abzuschliessen.  Ich arbeitete heute ausserplanmässig an der Dokumentation der Realisierungsphase, die ich auf Ende der Realisierungsphase verschoben hatte, weil es ständig Änderungen im bereits Dokumentierten Code gab. | |
| **Erfolge** | Die Implementierungsphase wurde erfolgreich abgeschlossen, einschliesslich der Fertigstellung der letzten Tests und dem Pushen der finalen Commits. Sowohl das Tracing als auch das Logging wurden schneller als erwartet abgeschlossen, was ermöglichte, dass die Tests bereits heute abgeschlossen werden konnten. | |
| **Misserfolge / Probleme und Lösungen** | Die Beschreibung der Tests wurde von „Unit Tests“ zu „Tests“ geändert, damit klar verständlich ist, dass auch Frontend Tests inklussive sind. | |
| **Hilfestellung und Begründung** |  | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Heute war ein sehr erfolgreicher und zufriedenstellender Tag, da ich mit dem Programmieren fertig wurde und somit im Code alle notwendigen Features implementiert wurden. Die frühere Fertigstellung des Tracings und Loggings sowie der Tests gibt mir die Möglichkeit, mich morgen intensiver mit der notwendigen Dokumentation zu befassen, die ich aufgrund konstanter Änderungen im Code auf den Schluss der Realisierungsphase verschoben hatte. | |

### Tag 7: Donnerstag, 11.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Tests schreiben * Automatisierte Unit Tests * Testprotokoll durchführen * Dokumentation – Realisieren * Dokumentation – Arbeitsjournal | | * Dokumentation – Realisieren * Automatisierte Unit Tests * Testprotokoll durchführen * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Der Zeitplan wurde durch die längere als erwartete Dauer der Dokumentationsarbeit beeinträchtigt, was dazu führte, dass ich später als geplant mit den automatisierten Tests beginnen musste. Diese Verspätung des Zeitplans lag daran, dass ich die Realisierungsphase in der Dokumentation ständig anpassen musste, da ich nicht wie im Zeitplan zuerst das Backend und dann das Frontend implementieren konnte, sondern an beiden ständige Anpassungen notwendig waren.  Ebenfalls habe ich diesen Teil der Dokumentation stark unterschätzt. | |
| **Erfolge** | Ich habe die Dokumentation der Realisierung strukturiert, indem ich den Ablauf der Anwendung nachverfolgte. Diese Methodik erwies sich als effektiv, um den Prozess klar und verständlich für Dritte darzustellen. | |
| **Misserfolge / Probleme und Lösungen** | Während der Durchführung der automatisierten Tests stiess ich auf Probleme, wo drei Tests fehlschlugen. Diese Tests waren nicht neu, sondern ältere, die durch Anpassungen an den Models und insbesondere den Mappers beeinträchtigt wurden.  Ich hatte diese Fehler in der Realisierungsphase nicht gemerkt, weil ich die neu implementierten Tests einzeln ausführte. Heute habe ich aber alle Tests im Projekt laufen gelassen und kam auf diese Fehler.  Eine geplante Korrektur dieser Fehler ist für den nächsten Tag vorgesehen. | |
| **Hilfestellung und Begründung** |  | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der heutige Tag war lehrreich hinsichtlich der Bedeutung einer gründlichen Dokumentation und des unerwarteten Auftretens von Problemen im älteren Code, wenn man neue Implementationen einführt, was die Notwendigkeit einer kontinuierlichen Überprüfung und Anpassung unterstreicht. Die Herausforderungen zeigten die Komplexität der Anwendungsintegration. Ich lernte, dass vor allem die Realisierungsphase recht schwer sein kann einzuplanen. | |

### Tag 8: Freitag, 12.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Testprotokoll durchführen * Vergleich IST/SOLL * Reflexion und Fazit * Dokumentation – Kontrollieren * Dokumentation – Auswerten * Dokumentation – Arbeitsjournal | | * Tests schreiben * Automatisierte Unit Tests * Testprotokoll durchführen * Vergleich IST/SOLL * Dokumentation – Kontrollieren * Reflexion und Fazit * Dokumentation – Auswerten * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Ich hatte am Freitag bei der Durchführung gemerkt, dass beim Testen manche der Tests noch fehlschlugen. Diese Tests mussten heute korrigiert werden. Der IST/SOLL Vergleich dauerte etwa eine halbe Stunde länger als erwartet, dies konnte ich aber durch das Dokumentieren des Kontrollierens ausgleichen. | |
| **Erfolge** | Ich konnte heute das Problem mit den Tests reparieren. Neben dem konnte ich heute das Testprotokoll durchführen und somit die ganze Kontrolle dokumentieren.  Die Phase Auswerten habe ich heute auch fast fertig gehabt, aber noch nicht zu meiner Dokumentation hinzugefügt, ich habe es nur stichwortartig gemacht. Das Dokumentieren dieser Phase wird wie nach Plan morgen passieren. | |
| **Misserfolge / Probleme und Lösungen** |  | |
| **Hilfestellung und Begründung** |  | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der erste Tag der Woche war produktiv, trotz des Stresses, dass die Unit Tests im Backend nicht funktionierten. Ich konnte das Problem innerhalb einer halben Stunde beheben und somit die Kontrollphase meiner Dokumentation abschliessen.  Nachdem konnte ich meine Auswertung stichwortartig auch für die morgige Dokumentation vorbereiten. Heute war ein allgemein guter Tag, mit gutem Fortschritt. Viele der Aufgaben waren schneller erledigt als gedacht, was mir erlaubte, ein wenig vor der geplanten Zeit zu schaffen. Dies wird mir erlauben, meine IPA in den letzten Tagen besser durchzugehen und somit ein besseres Produkt zu liefern. | |

### Tag 9: Montag, 15.04.2024

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Reflexion und Fazit * Kriterien durchgehen * Dokumentation – Auswerten * Dokumentation – Arbeitsjournal | | * Dokumentation – Auswerten * Reflexion und Fazit * Kriterien durchgehen * Dokumentation – Arbeitsjournal |
| **Ausserplanmässige Arbeiten** | Da ich gestern früher mit der Reflexion und dem Fazit begonnen hatte, konnte ich heute die Aufgabe entsprechend eine halbe Stunde früher abschliessen. | |
| **Erfolge** | Heute konnte ich die letzte meiner IPERKA-Phasen abschliessen. Nun kann ich mich gezielt auf die Qualität und Vollständigkeit meiner Dokumentation konzentrieren. | |
| **Misserfolge / Probleme und Lösungen** |  | |
| **Hilfestellung und Begründung** |  | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der heutige Halbtag verlief gut. Ich konnte die letzte Phase der IPERKA-Vorgehensweise abschliessen, was bedeutet, dass soeben alles dokumentiert ist. Dies bedeutet jedoch nicht, dass ich die Dokumentation bereits in dieser Form abgeben kann. Während des Projekts habe ich mich grösstenteils nicht auf die Rechtschreibung konzentriert. Aus diesem Grund werde ich die Dokumentation morgen eingehend überprüfen, um sicherzustellen, dass sie keine Fehler enthält. Ebenfalls muss ich den Code in einem Anhang eines Word-Dokuments kopieren und kennzeichnen, was von mir im Rahmen dieser IPA verfasst wurde. | |

### Tag 10: Dienstag, 16.04.2024 – Halbtag

|  |  |  |
| --- | --- | --- |
| **Aufgaben gemäss Zeitplan** | | **Heutige Arbeiten** |
| * Kriterien durchgehen * Korrekturlesen und Abgabe * Dokumentation – Arbeitsjournal | | * Dokumentation – Arbeitsjournal * Kriterien durchgehen * Korrekturlesen und Abgabe |
| **Ausserplanmässige Arbeiten** |  | |
| **Erfolge** | Die erfolgreiche Abgabe der IPA war der Höhepunkt des Tages. Ich konnte alle erforderlichen Anhänge fertigstellen und die Dokumentation grammatisch korrigieren. Zusätzlich ersetzte ich einige schmierige Screenshots durch klarere SVG-Bilder, was die Qualität der visuellen Darstellung verbesserte. | |
| **Misserfolge / Probleme und Lösungen** |  | |
| **Hilfestellung und Begründung** | Zur Korrektur und Verbesserung meiner Dokumentation nutzte ich zwei Online-Tools: ChatGPT für die Überprüfung der Fachsprache und languagetool.org für die Grammatikkorrektur. Diese Werkzeuge halfen dabei, die Qualität und Genauigkeit der finalen Dokumentation sicherzustellen. | |
| **Überzeit und Begründung** |  | |
| **Persönliche Reflexion** | Der letzte Tag der IPA war stressig, aber letztendlich sehr erfolgreich. Das mehrfache Durchlesen und Korrigieren der Dokumentation war anstrengend, aber notwendig, um ein qualitativ hochwertiges Produkt abzugeben. Ich bin stolz auf das fertige Produkt und habe viel aus dieser Erfahrung gelernt. | |

### Tag 11: Mittwoch, 17.04.2024

Teil 2 Individueller praktischer Teil

## Kurzfassung

### Ausgangssituation

In der Anwendung OVWEB ist bereits die Erstellung eines Angebots für eine Todesfallversicherung möglich. Allerdings fehlt die Option, Angebote mit verschiedenen Versicherungssummen zu vergleichen. Das Ziel dieser IPA besteht darin, sowohl dem Berater als auch dem Kunden einen benutzerfreundlichen Vergleich anzubieten und dessen Erstellung zu ermöglichen.

### Projektvorgehen

Die Durchführung des Projekts folgt der IPERKA-Methode (siehe Kapaitel 7). Nach einer eingehenden Informations- und Planungsphase, in der der aktuelle Zustand erfasst und der gewünschte Zustand definiert wurde, kann nun mit einem klaren Verständnis der Ziele die Implementierungsphase beginnen. Während der Implementierungsphase erfolgen Anpassungen am Controller und am Service. Um diese Änderungen zu implementieren, werden ebenso die Modelle, DTOs und die Datenbank modifiziert. Im Frontend-Bereich des Projekts wird die Detail- und die Erstellungsseite angepasst. Für die Erstellungsseite sind neue Eingabefelder sowie Validierungen notwendig. Die Detailseite wird derart überarbeitet, dass sie übersichtlich bis zu drei Angebote gleichzeitig darstellen kann.

### Ergebnis

Die Einführung des Prämienvergleichs für Todesfallversicherungen in der OVWEB-Anwendung hat zu signifikanten Verbesserungen geführt. Die Berater können nun bis zu drei Versicherungsangebote gleichzeitig einsehen und vergleichen. Die Integration des Rechenkerns ermöglicht eine effektive Validierung der eingegebenen Daten und sorgt für die direkte Anzeige von Fehlermeldungen im Frontend. Diese Funktionalitäten erleichtern es den Beratern, genaue und vergleichende Analysen vorzunehmen und optimale Versicherungsoptionen ihren Klienten vorzuschlagen.

## Projektvorgehen

Die IPERKA-Methode ist ein systematischer Ansatz zur Problemlösung und Projektplanung.   
Sie gliedert den Prozess in sechs Phasen:   
Informieren, Planen, Entscheiden, Realisieren, Kontrollieren und Auswerten.   
Diese Methode hilft dabei, komplexe Aufgaben strukturiert anzugehen, Ziele klar zu definieren und die Umsetzung effektiv zu gestalten.

**Informieren**:

In der ersten Phase geht es darum, alle relevanten Informationen zu sammeln und zu bewerten. Dies beinhaltet das Verständnis der Aufgabe, das Identifizieren von notwendigen Ressourcen und das Analysieren des jetzigen Zustandes sowie der Systemumgebung.

**Planen**:

Nachdem alle Informationen gesammelt wurden, wird in dieser Phase ein detaillierter Plan erstellt. Dabei werden Ziele definiert (Soll-Zustand), erforderliche Schritte zur Zielerreichung ausgearbeitet, Diagramme erstellt, Mockups angefertigt und ein Testkonzept festgelegt.

**Entscheiden**:

Basierend auf den gesammelten Informationen und dem erarbeiteten Plan wird eine Entscheidung über die optimale Implementierung getroffen. Die Auswahl erfolgt anhand eines Kriterienkatalogs.

**Realisieren**:

Die Realisierungsphase umfasst die aktive Umsetzung des Plans. In dieser Phase werden die geplanten Schritte ausgeführt und kontinuierlich dokumentiert, um den Fortschritt nachvollziehbar zu machen.

**Kontrollieren**:

Nach der Umsetzung ist es wichtig, den Fortschritt zu überprüfen. Dies geschieht anhand der zuvor definierten Tests, um sicherzustellen, dass alles wie geplant funktioniert und alle Ziele erreicht wurden.

**Auswerten**:

Nach Abschluss der Arbeit wird der gesamte Prozess ausgewertet. Diese Reflexion beinhaltet die Bewertung des erreichten Ergebnisses im Vergleich zu den gesetzten Zielen. Ebenfalls werden gesammelte Erfahrungen analysiert, um sowohl die Stärken als auch die Verbesserungspotenziale des Projekts zu bewerten.

## Informieren

### Ziele der Aufgabenstellung

Das Ziel dieser Aufgabe besteht darin, dem Berater einen Prämienvergleich mit bis zu drei Angeboten für Todesfallversicherungen zu präsentieren. Zudem soll dem Berater eine Fehlermeldung angezeigt werden, wenn die für die Erstellung des Angebots eingegebenen Daten gegen die Geschäftsregeln verstossen.

### Vorgaben und Anforderungen

Der Berater muss in der Lage sein, bis zu drei unterschiedliche Versicherungssummen einzugeben. Auf der Detailseite soll es ihm möglich sein, alle drei Angebote gleichzeitig zu betrachten, um einen Vergleich anstellen und die Angebote präsentieren zu können. Die für das Angebot eingegebenen Daten müssen durch den Rechenkern der vorhandenen Angebotssoftware validiert werden. Sollten dabei Fehler auftreten, müssen diese im Frontend sichtbar gemacht werden, sodass der Berater sie überprüfen und beheben kann.

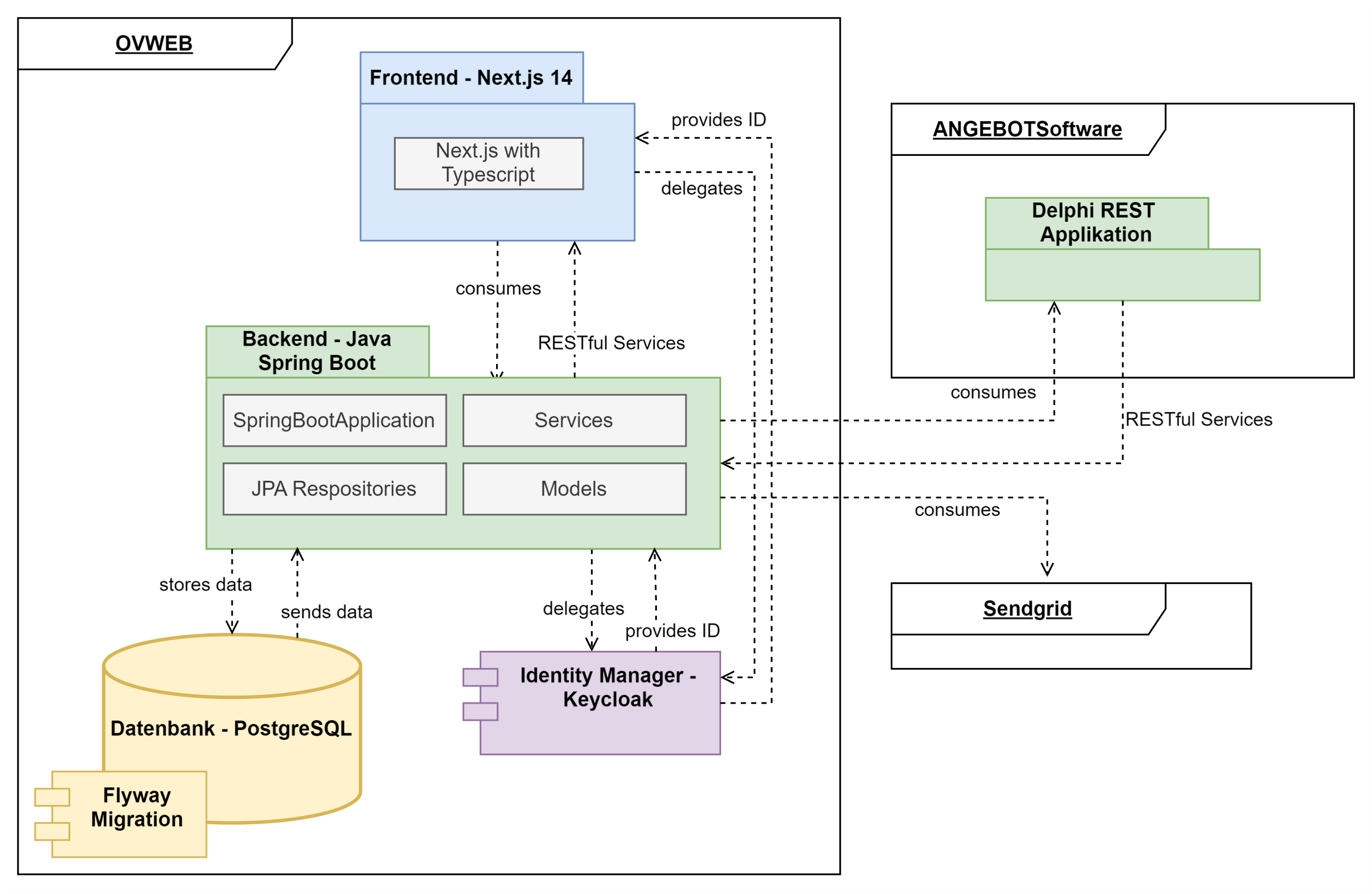
### Ist-Zustand

Aktuell ermöglicht die OVWEB-Anwendung die Erstellung einer Todesfallversicherung über ein Formular, welches anschliessend im Backend der OVWEB-Anwendung gespeichert wird, unabhägig davon, ob sie die Business-Rules verletzt oder nicht.   
Nach der Erstellung erfolgt eine Weiterleitung im Frontend zur Detailseite mit der Offerten-ID. Dort wird erneut eine Abfrage an das Backend gesendet, um die Daten mithilfe des Rechenkerns der bisherigen ANGEBOTSoftware zu verarbeiten. Das Ergebnis dieser Berechnung, sei es eine Offerte oder eine Fehlermeldung, wird dann auf der Detailseite im Frontend dargestellt.

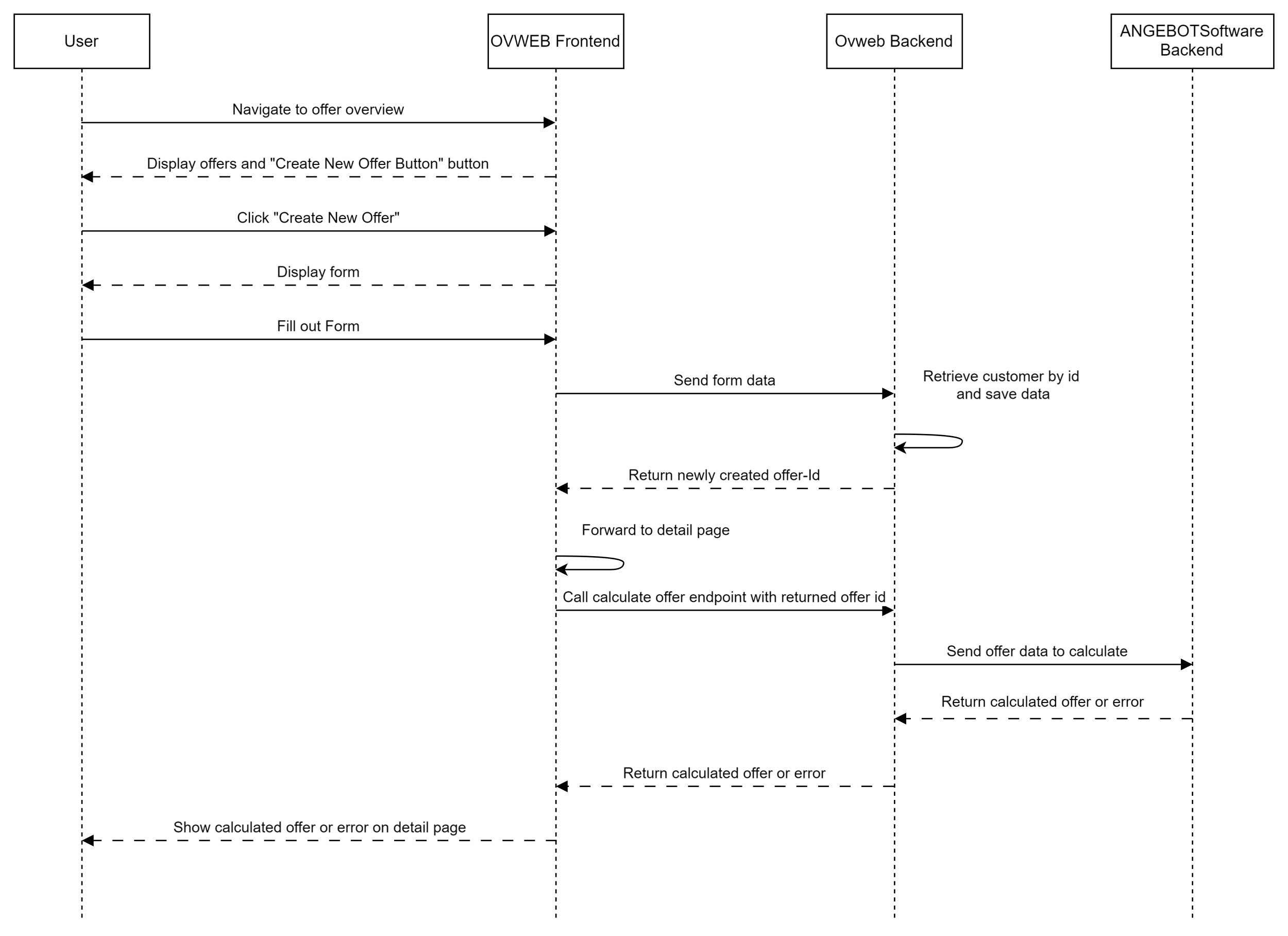
Zusätzlich gibt es eine Übersichtsseite für Offerten, auf der alle bislang erstellten Angebote aufgelistet sind. Ein Klick auf eines dieser Angebote führt mittels des beschriebenen Prozesses zurück auf die Detailseite des spezifischen Angebots.

#### Projekt-Überblick

Das nachfolgende Diagramm bietet eine klare Darstellung der OVWEB-Anwendung sowie ihrer Abhängigkeiten.



Aufgrund der Komplexität des aktuellen Prozesses zur Erstellung einer Offerte wurde ein Sequenzdiagramm gefertigt, um diesen Vorgang vereinfacht zu visualisieren.

Das Diagramm dient der Übersicht und beschreibt die wichtigsten Prozesse.

#### Frontend

*Kommunikation zum Backend*

Die Kommuniztion zwischen dem Frontend und dem Backend erfolgt unter Benutzung des Client-API-Packages, welches automatisch von OpenAPI Maven Plugin basierend auf der Endpunktannotation im Backend generiert wird. Somit wird die typensichere Integration zwischen dem OVWEB Frontend und OVWEB Backend sichergestellt.

*Server Actions*

Im Rahmen des OVWEB Frontends werden Server Actions verwendet, um mit der OVWEB-Backend-API zu interagieren. Diese Server Actions ermöglichen es, serverseitige Programmierlogik direkt im Frontend zu verwenden, was die Entwicklung effizienter und flexibler gestaltet. Durch die Nutzung dieser Technik können Daten abgerufen, verarbeitet und an das Backend gesendet werden, ohne dass der Browser zwischenspeichern muss, was die Performance und die Sicherheit der Plattform verbessert.

Storybook

Storybook wird als eine zentrale Bibliothek für UI-Komponenten und Seiten verwendet. In das Storybook werden sämtliche entwickelten Komponenten und Seiten eingetragen, sodass sie unabhängig vom Hauptsystem visualisiert werden können. Dies ermöglicht es, dass auch Personen ohne technischen Hintergrund die UI-Elemente betrachten und beurteilen können, ohne dass die gesamte Applikation gestartet werden muss.

**A white background with red and blue text

Description automatically generatedOffer Dashboard**

Das Offerten-Dashboard bietet eine Übersicht über alle vorhandenen Offerten. Im Rahmen dieser IPA erfolgen keine Anpassungen an dieser Seite. Dennoch ist sie von zentraler Bedeutung, da über sie der Zugang sowohl zur Detailseite als auch zur Erstellungsseite für Offerten möglich ist. Zum Erstellen einer Offerte wird der Button „Neue Versicherung“ verwendet. Um eine bestehende Offerte zu berechnen und detailliert einzusehen, klickt man auf den Text „View Details“.

**A white paper with red and blue lines

Description automatically generatedOffer Creation**

Die Erstellungsseite ermöglicht das Anlegen einer Offerte. Das Formular integriert eine frontend-seitige Eingabevalidierung mittels Zod. Durch Betätigung des Buttons „Speichern“ wird die Server-Aktion „create-offer“ ausgelöst, wobei die Offerte im Backend gespeichert wird, unabhängig davon, ob sie die Business-Rules verletzt oder nicht.

**A screenshot of a computer

Description automatically generatedOffer Details – Erfolgreiche Berechnung der Offerte**

Beim Aufruf der Offertendetailseite wird die entsprechende Offerte mittels Rechenkern der ANGEBOTSoftware berechnet. Die Berechnungsdetails werden im Frontend dann auf dieser Seite angezeigt.

**Offer Details – Fehlgeschlagene Berechnung der OfferteA screenshot of a computer

Description automatically generated**

Beim Aufruf der Offertendetailseite wird die entsprechende Offerte mittels Rechenkern der ANGEBOTSoftware berechnet. Falls bei der Berechnung eine Verletzung der Business-Rules stattfindet, wird der vom Rechenkern generierte Fehler auf dieser Seite angezeigt.

#### 

#### Backend

Die Klassendiagramme sind dazu bestimmt, einen Überblick zu verschaffen und das Verständnis der Systemarchitektur sowie der Beziehungen zwischen den verschiedenen Klassen in Bezug auf Offerten zu verbessern.

*Für die Verwaltung der Offerten ist der Service und der Controller mit der Benennung LifeInsuranceRequest zuständig. Dies beinhaltet die CRUD-Methoden.*

*Für die Berechnung der Offerten ist der Service und der Controller mit der Benennung LifeInsuranceCalculation zuständig.*

**LifeInsuranceRequest –> Verwaltung der Offerten**

**LifeInsuranceCalculation -> Berechnung der Offerten**

Im Rahmen dieser IPA wird nur die Erstellung sowie die Berechnung einer Offerte in Detail betrachtet.

Die Offerten werden zusammen mit den Kundendaten in der Datenbank gespeichert, ohne dass eine Berechnung stattfindet. Dies geschieht, weil die Kundendaten nach der Erstellung der Offerte unveränderlich sein müssen.

**Controller und Service, Verwaltung der Offerten**

Folgend sind die Controller und Services für die Verwaltung der Offerten aufgeführt. Diese sind im Rahmen der IPA für die Speicherung der Offerten verantwortlich. Sie umfassen die standardmässigen CRUD-Methoden sowie einen Mapper.A screenshot of a computer screen

Description automatically generated

**Model und DTOs, Verwaltung der Offerten**

Das LifeInsuranceRequestPostDto dient dazu, die im Frontend eingegebenen Daten via Controller zur Offertenerstellung zu empfangen. Dabei wird die Kunden-ID übertragen und nicht die vollständigen Kundendaten sowie die Berechnungsdaten für die Offerte.

A screenshot of a computer program

Description automatically generated

Die Kunden-ID wird in der Variable customerId gespeichert und anschliessend zusammen mit den Offertendaten aus dem DTO an den Service weitergeleitet. Dort erfolgt die Abfrage der Kundendaten aus der Datenbank unter Verwendung der zuvor genannten ID.

Zusätzlich wird die Sponsor-ID abgespeichert. Diese ist ein Fremdschlüssel für die User-Tabelle, welche unsere registrierten Berater beinhaltet. Diese Sponsor-ID wird mittels JWT, welches bei der CRUD Operation für die Authentifizierung notwendig ist, aus der Datenbank ausgelesen.

Zusammen mit den Offertendaten aus dem DTO und der Sponsor-ID werden die Kundendaten in diesem Modell zugeordnet. Dafür wird die Methode mapToLifeInsuranceRequest genutzt, die im LifeInsuranceRequestService implementiert ist.

A screenshot of a computer

Description automatically generated

Diese Daten werden anschliessend über das Repository in der Datenbank gespeichert, wobei die automatisch generierte Offerten-ID zurückgegeben und an das Frontend weitergeleitet wird. Mit dieser ID wird dann die entsprechende Detailseite aufgerufen.

Die Detailseite initiiert mit der Offerten-ID einen Aufruf des Endpunkts  
”/get-offer-details/{id}”, der die Offerte mithilfe der externen ANGEBOTSoftware berechnet. Diese Berechnung kann, wie in Kapitel 8.3.2 dargestellt, bei einer Verletzung der Business-Rules auch einen Fehler zurückgeben.

**Controller und Service, Berechnung der Offerte**

A screenshot of a computer

Description automatically generated

Im Controller wird der Request zusammen mit der Offerten-ID empfangen. Anhand dieser ID wird anschliessend die Methode calculateOffer im Service aufgerufen. In diesem Service werden die in der Datenbank gespeicherten Daten abgerufen und mittels der Methode mapToFosoftRequest so aufbereitet, dass sie die erforderliche Struktur für die API der ANGEBOTSoftware aufweisen.

Anschliessend wird die Struktur des Requests und des Responses dargelegt.

**Request:**

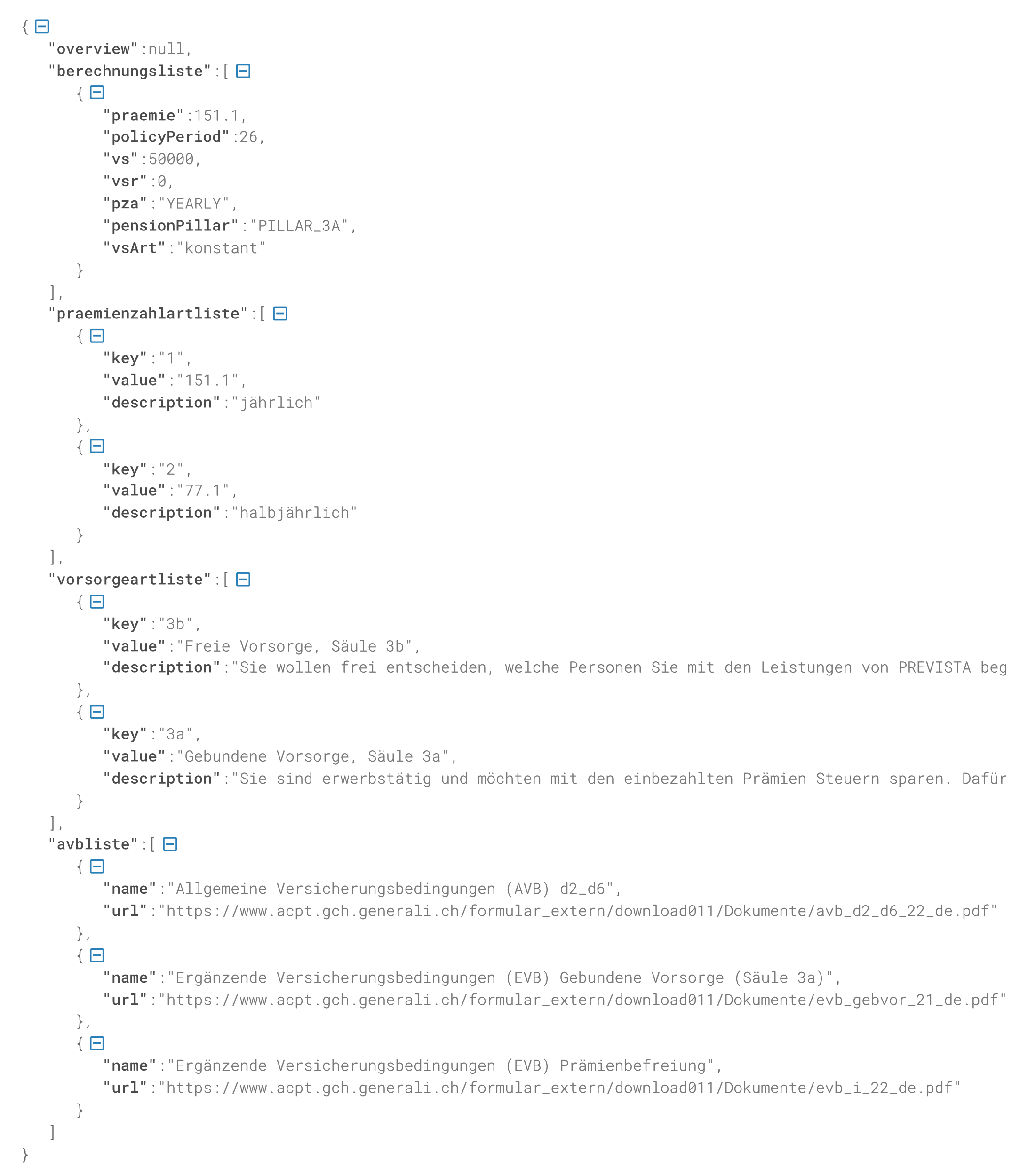
Folgend befindet sich ein Beispiel für ein Request an dem Endpoint D2.

A screenshot of a computer code

Description automatically generated

**Response – Erfolgreiche Berechnung:**

Folgend dargestellt ist die Response der ANGEBOTSoftware im Falle einer erfolgreich berechneten Offerte.



Diese Response könnte jedoch im Falle, dass die Business-Rules verletzt werden, auch einen Fehler aufweisen. Dies würde wie folgt aussehen:

**Response – Verletzung der Business-Rules:**

A computer screen shot of a message

Description automatically generated

Die Response wird vom OVWEB-Backend an das Frontend zurückgesendet und dort für den Berater visualisiert.

#### ERM-Datenbankdiagramm

A screenshot of a computer

Description automatically generated

*User-Table*

In der User-Tabelle werden die Berater und deren Daten abgespeichert.

*Customer-Table*

In der Customer-Tabelle werden die Kunden der Berater und deren Informationen abgespeichert.

*Adress-Table*

Diese Tabelle dient der Speicherung von Adressen der Kunden und der Berater.

*Task-Table*

Die Task-Tabelle ist für die Speicherung der Pendenzen implementiert.

***Life-Insurance-Request-Table***

Die Life-Insurance-Request-Tabelle dient zur Speicherung von Todesfallversicherungen. Im Rahmen dieser IPA ist diese die wichtigste Tabelle.

*Flyway Schema History-Table*

Diese Tabelle wird von Flyway automatisch generiert und dient der Versionierung der Datenbankschemata. Sie ermöglicht die Nachverfolgung und Verwaltung von Schema-Migrationen.

#### Systemgrenzen

Diese Arbeit zielt auf die Erstellung, Berechnung, Validierung und Anzeige einer Offerte oder eines Offerten-Bundles. Daher gehört das ANGEBOTSoftware, spezifisch den Rechenkern davon, als Abhängigkeit zu dieser Aufgabe. Die Generali stellt die API via Webservices zur Verfügung.

Die ANGEBOTsoftware ist eine bereits etablierte Applikation, deren Modifikation oder Anpassung nicht möglich ist. Der Zugriff erfolgt über den **Endpoint D2**, der für die Berechnung und Überprüfung der Todesfallversicherungen zuständig ist. Die Daten für Anfragen und Antworten sind in Kapitel 8.3.3 dokumentiert.

Es ist wichtig zu erwähnen, dass ein Email-Service auf Basis von SendGrid und Authentifizierungservice auf Basis von Keycloak auch Teil des Projektes sind. Die mit diesen Technologien verbundenen Implementationen werden aber im Rahmen dieser IPA nicht bearbeitet.

### User Stories und Akzeptanzkriterien

Im nachfolgenden Kapitel erfolgt eine detaillierte Analyse und Aufschlüsselung der Anforderungen aus der Aufgabenstellung. Dies wird durch die Definition von User Stories erreicht, die anhand von Akzeptanzkriterien überprüft werden können.

#### Funktionale Anforderungen

|  |
| --- |
| **User Story 1: Erstellung einer Offerte mit mehreren Versicherungssummen** |
| **Als** Versicherungsberater  **möchte ich** die Möglichkeit haben, eine Offerte mit bis zu drei verschiedenen Versicherungssummen hinzufügen zu können,  **damit** ich für meinen Kunden einen Vergleich anbieten kann. |
| **Akzeptanzkriterien** |
| **AC 1.1:** Ein Berater findet auf der Seite zur Erstellung von Offerten auf drei Eingabefelder für die Versicherungssumme. |
| **AC 1.2:** Ein Berater ist in der Lage, bis zu drei Versicherungssummen einzugeben um somit ein Offerten-Bundle zu erstellen. |

|  |
| --- |
| **User Story 2: Offertenvergleich** |
| **Als** Versicherungsberater  **möchte ich** die Möglichkeit haben, einen Vergleich zwischen den zuvor erstellten Offerten einsehen zu können,  **damit** ich diese gemeinsam mit dem Kunden vergleichen kann. |
| **Akzeptanzkriterien** |
| **AC 2.1:** Wenn der Berater eine Offerte mit mehreren Versicherungssummen erstellt hat, sollte ihm eine Übersicht über die drei Berechnungen zur Verfügung stehen, damit er diese gemeinsam mit dem Kunden vergleichen kann. |

|  |
| --- |
| **User Story 3: Offerten-Validierung** |
| **Als** Versicherungsberater  **möchte ich** sicherstellen, dass ich keine fehlerhaften Offerten erstellen kann,  **um** dem Kunden vor Ort sofort eine korrekte Offerte präsentieren zu können. |
| **Akzeptanzkriterien** |
| **AC 3.1:** Wenn ein Berater eine Offerte mit fehlerhaften Daten erstellt, sollten diese nicht in der Datenbank gespeichert werden. |
| **AC 3.2:** Wenn ein Berater eine Offerte mit fehlerhaften Daten erstellt, sollte eine Weiterleitung auf die Detailseite unterbleiben. |
| **AC 3.3:** Erstellt ein Berater eine Offerte mit fehlerhaften Daten, sollte eine Benutzerinteraktion erfolgen, die genau erläutert, was an der Offerte fehlerhaft ist. |

#### Nichtfunktionale Anforderungen

|  |
| --- |
| **User Story 4: Benutzerfreundlichkeit der UI** |
| **Als** Versicherungsberater  **möchte ich**, dass meine Offerten leicht miteinander vergleichbar sind,  **damit** der Kunde sie auch verstehen kann. |
| **Akzeptanzkriterien** |
| **AC 4.1:** Die verschiedenen Offerten sind leicht miteinander zu vergleichen. Das bedeutet, dass die wichtigen Informationen, wie die Prämien, auf gleicher Höhe angeordnet und klar ersichtlich sind. |
| **AC 4.2:** Fehlermeldungen bei der Erstellung einer Offerte sind eindeutig dargestellt und leicht verständlich. |

|  |
| --- |
| **User Story 5: Observability-Logging** |
| **Als** Systemadministrator  **möchte** ich detaillierte Logs über fehlgeschlagene Offertenerstellungen sowie Berechnungen erhalten,  **um** mögliche Probleme bei der Erstellung und bei der Berechnung analysieren und beheben zu können. |
| **Akzeptanzkriterien** |
| **AC 5.1:** Das Logging innerhalb des Systems erfolgt strukturiert. Die Log-Nachrichten weisen ein einheitliches Format auf, und alle relevanten Daten sind erfasst. |
| **AC 5.2:** Die Implementierung des Loggings ist systemübergreifend in allen Komponenten des Systems sichergestellt, um eine nachvollziehbare Protokollierung zu gewährleisten. |
| **AC 5.3:** Alle Log-Einträge enthalten die erforderlichen Informationen, um das Verhalten des Systems nachzuvollziehen und bei Bedarf eine Fehlersuche durchführen zu können. |
| **AC 5.4:** Sensible Daten, insbesondere Passwörter und personenbezogene Informationen, werden zu keinem Zeitpunkt in den Logdateien gespeichert. |

|  |
| --- |
| **User Story 6: Verbesserung der Nachverfolgbarkeit durch Trace-IDs** |
| **Als** Entwickler  **möchte ich** die Möglichkeit haben, jede Systemanfrage mittels einer einzigartigen Trace-ID in Grafana nachverfolgen zu können,  **um** eine effiziente Fehlerbehebung zu ermöglichen. |
| **Akzeptanzkriterien** |
| **AC 6.1:** Bei jeder Anfrage im Frontend wird eine Trace-ID generiert, die dann an das Backend weitergeleitet wird. |
| **AC 6.2:** Die generierte Trace-ID wird konsequent in den Logs aller beteiligten Dienste festgehalten. Dies verbessert die Nachverfolgbarkeit und erleichtert die Diagnose durch das Logging. |
| **AC 6.3:** Die lückenlose Dokumentation der Trace-IDs ermöglicht die Erstellung detaillierter Diagramme der Service-Aufrufe, welche die Interaktionen und den Datenfluss zwischen den verschiedenen Diensten visualisieren. |

|  |
| --- |
| **User Story 7: Testabdeckung für neue OVWEB-Systemfunktionen** |
| **Als** Entwickler  **möchte ich**, dass jede neue Funktion, egal ob im Frontend oder im Backend, durch automatisierte Tests abgesichert ist,  **um** die Zuverlässigkeit und Funktionsfähigkeit des Systems zu garantieren und  Fehler frühzeitig identifizieren und beheben zu können. |
| **Akzeptanzkriterien** |
| **AC 7.1:** Die Testfälle umfassen sowohl positive als auch negative Szenarien sowie Tests mit Null- und Nicht-Null-Skalarwerten. |
| **AC 7.2:** Für neue Funktionen, die effektiv und effizient durch Unit-Tests geprüft werden können, werden keine Integrationstests implementiert. |
| **AC 7.3:** Alle neu implementierten Funktionalitäten müssen getestet werden, sowohl im Frontend als auch im Backend. |

|  |
| --- |
| **User Story 8: Modularität und Wiederverwendbarkeit der UI-Komponenten** |
| **Als** Entwickler  **möchte ich**, dass die neuen Komponenten des Systems modular  aufgebaut und dokumentiert sind,  **damit** sie wiederverwendbar sind. |
| **Akzeptanzkriterien** |
| **AC 8.1:** Komplexe Komponenten sind aus kleineren, wiederverwendbaren Komponenten zusammengebaut. |
| **AC 8.2:** Diese Elemente müssen in dem Storybook dokumentiert werden. |

|  |
| --- |
| **User Story 9: Erfassung des Designs in Figma** |
| **Als** Entwickler  **möchte ich**, dass alle neuen Seiten in Figma erfasst sind,  **damit** ich sicherstellen kann, dass das Design konsistent bleibt und alle Entwicklungsanforderungen visualisiert und zugänglich sind. |
| **Akzeptanzkriterien** |
| **AC 9.1:** Alle Seitenentwürfe müssen in dem spezifizierten Figma-Projektfile vorliegen, bevor die Entwicklung beginnt. |
| **AC 9.2:** Alle Seitenentwürfe müssen die Figma-Farbpalette verwenden, um Designkonsistenz zu gewährleisten. |

## Planen

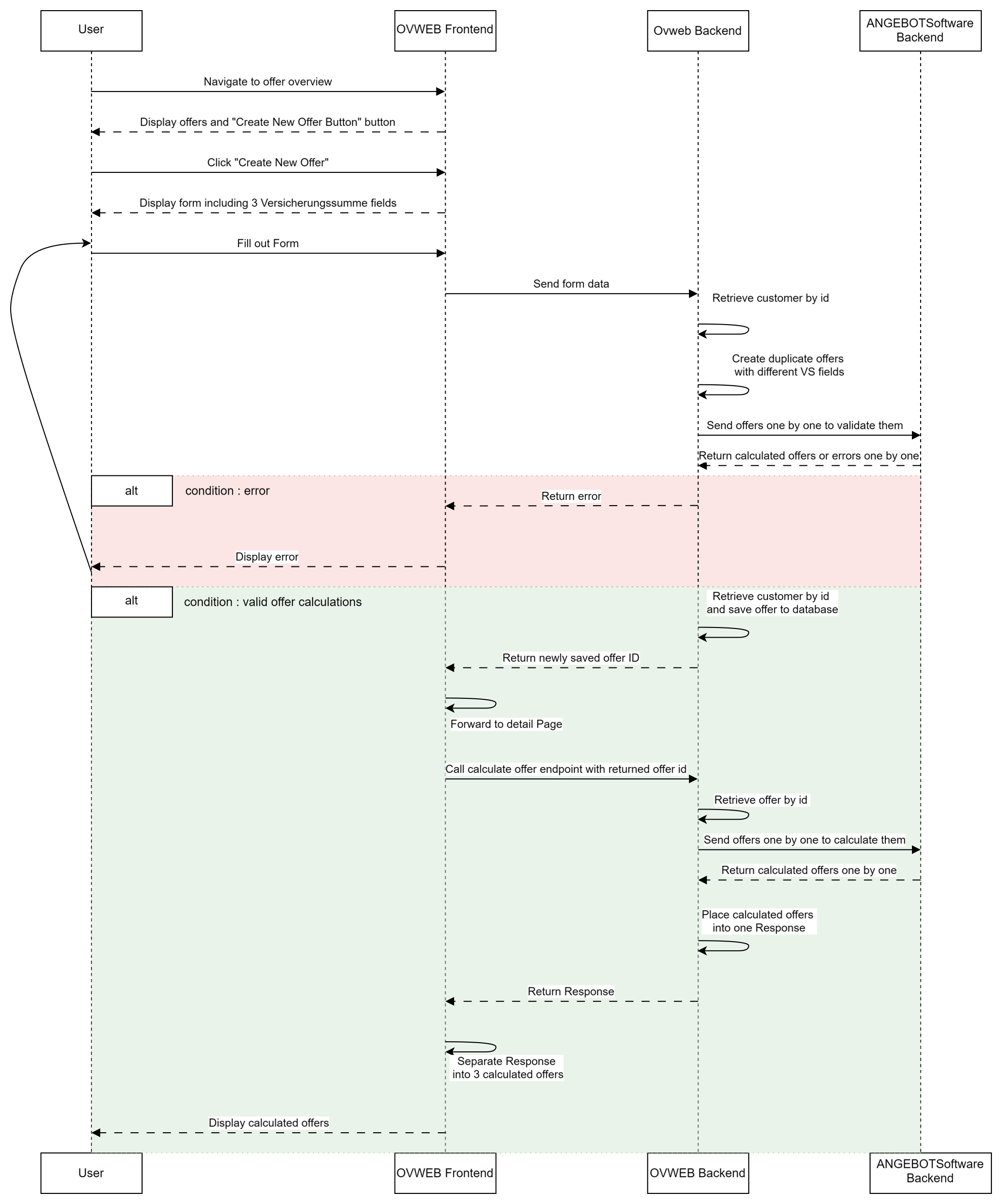
### Soll-Zustand

Der Soll-Zustand sollte es dem Berater erlauben, maximal drei Versicherungssummen einzugeben, um ein Offerten-Bundle zu erstellen und somit einen Vergleich der Prämien dem Kunden zu präsentieren.

Ebenfalls sollte eine Validierung bei der Erstellung des Offerten-Bundles passieren.

Um dies zu erreichen, werden grösstenteils Komponenten verändert und verwendet, die schon existieren, es werden aber auch neue erstellt.

Um die Verständlichkeit des neuen Prozesses zu erhöhen, wurde das in Abschnitt 8.3.1 dargestellte Diagramm entsprechend den geplanten Änderungen aktualisiert.



#### Frontendänderungen

In Kapitel 9.2 find Sie die dazugehörigen Mockups. Die neuen Komponenten werden den Mockups entsprechen.

**Storybook Anpassungen**

Alle neuen Komponenten müssen im Storybook erfasst werden.

**Tracing**

Für jedes Request wird eine Tracing-ID generiert.

**Offer Creation Page:**

* Anpassungen an der Form, um zwei neue Versicherungssumme-Felder hinzuzufügen
* Das Empfangen eines Fehlers muss möglich sein
* Das Anzeigen des Fehlers muss implementiert werden

**Offer Detail Page:**

* Ermöglichen maximal drei Offerten auf einmal zu empfangen (in einer Response)
* Diese drei Offerten gleichzeitig anzeigen
* Neue UI-Elemente müssen einfach zu lesen und zu vergleichen sein, so wie auch benutzerfreundlich

#### Backendänderungen

**Datenbankänderungen**

* Es braucht zwei neue Felder für die neuen Versicherungssummen in der Offerten Tabelle, dies wird durch eine Flyway Migration ausgeführt. Das bestehende Feld „vs“ wird zu „vs1“ umbenannt

**A white background with black text

Description automatically generated**

**Controller- und Serviceänderungen**

* Das bestehende Service und der Controller müssen so angepasst werden, sodass diese die Offerte zuerst validieren, bevor diese abgespeichert wird. Die Methode dafür wird im LifeInsuranceCalculationService implementiert. Da sie in direktem Kontakt mit der externen API ist, wird aber im LifeInsuranceRequestService aufgerufen, da diese Methode nicht für die Erstellung zuständig ist, sondern für die Validierung.
* Das bestehende Service und der Controller müssen bis zu drei Offerten empfangen und bearbeiten können.
* Tracing und Logging müssen implementiert werden.
* Die Mapping-Methoden müssen entsprechend angepasst werden.

Für die Validation muss ein Call am Rechenkern der ANGEBOTSoftware gemacht werden. Der gleiche Endpunkt wird benutzt, wie bei der Erstellung einer Offerte. Aus diesem Grund kann in der neuen Validierungsmethode die schon bestehende Methode calculateOffer aus dem LifeInsuranceCalculationController benutzt werden.

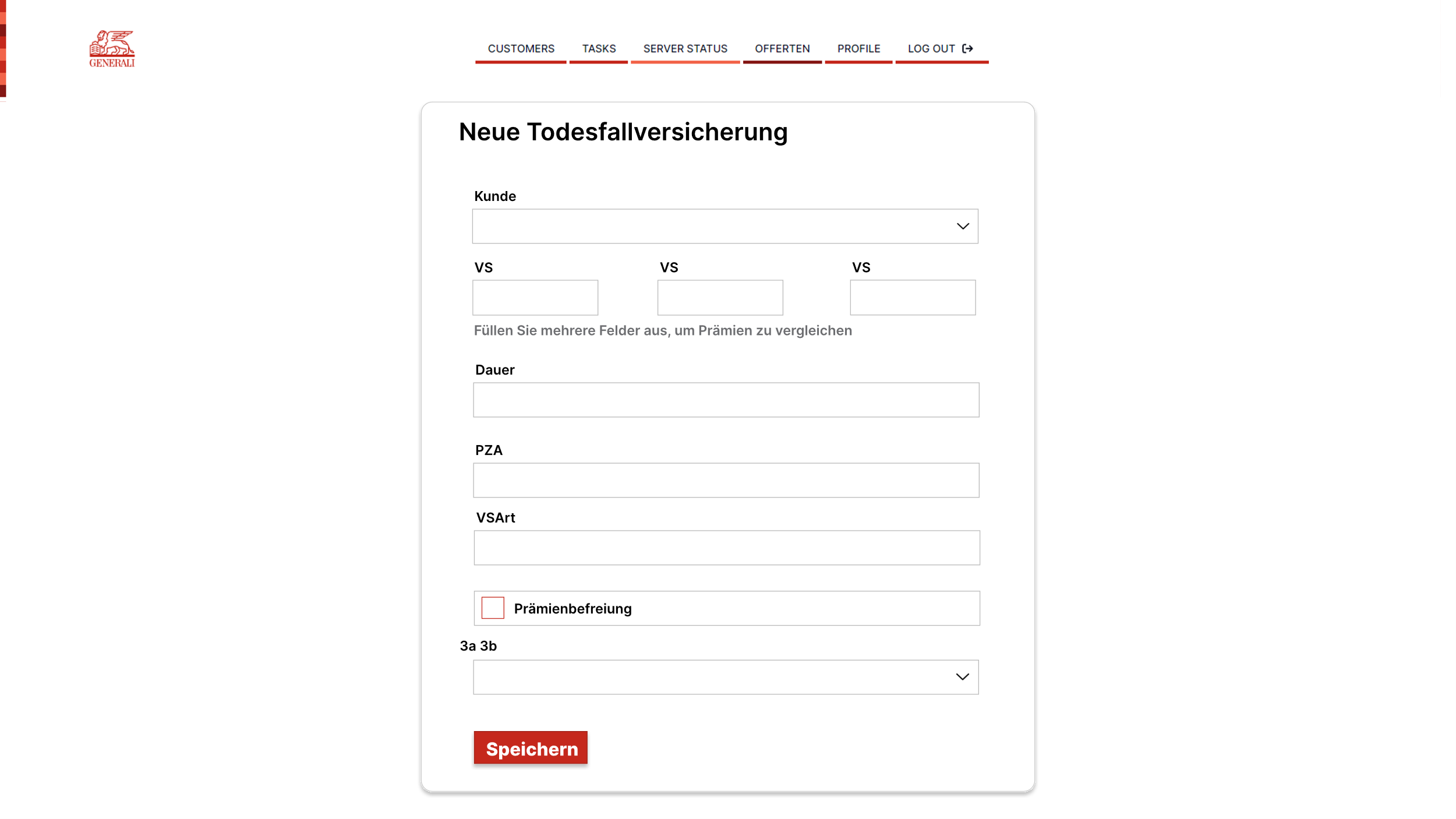
**Model und DTO Änderungen**

* Die Models und DTOs müssen so angepasst werden, dass sie drei Versicherungssummen handeln können.
* Die Models und DTOs werden so angepasst, dass Sie entsprechend im Frontend validiert werden können.

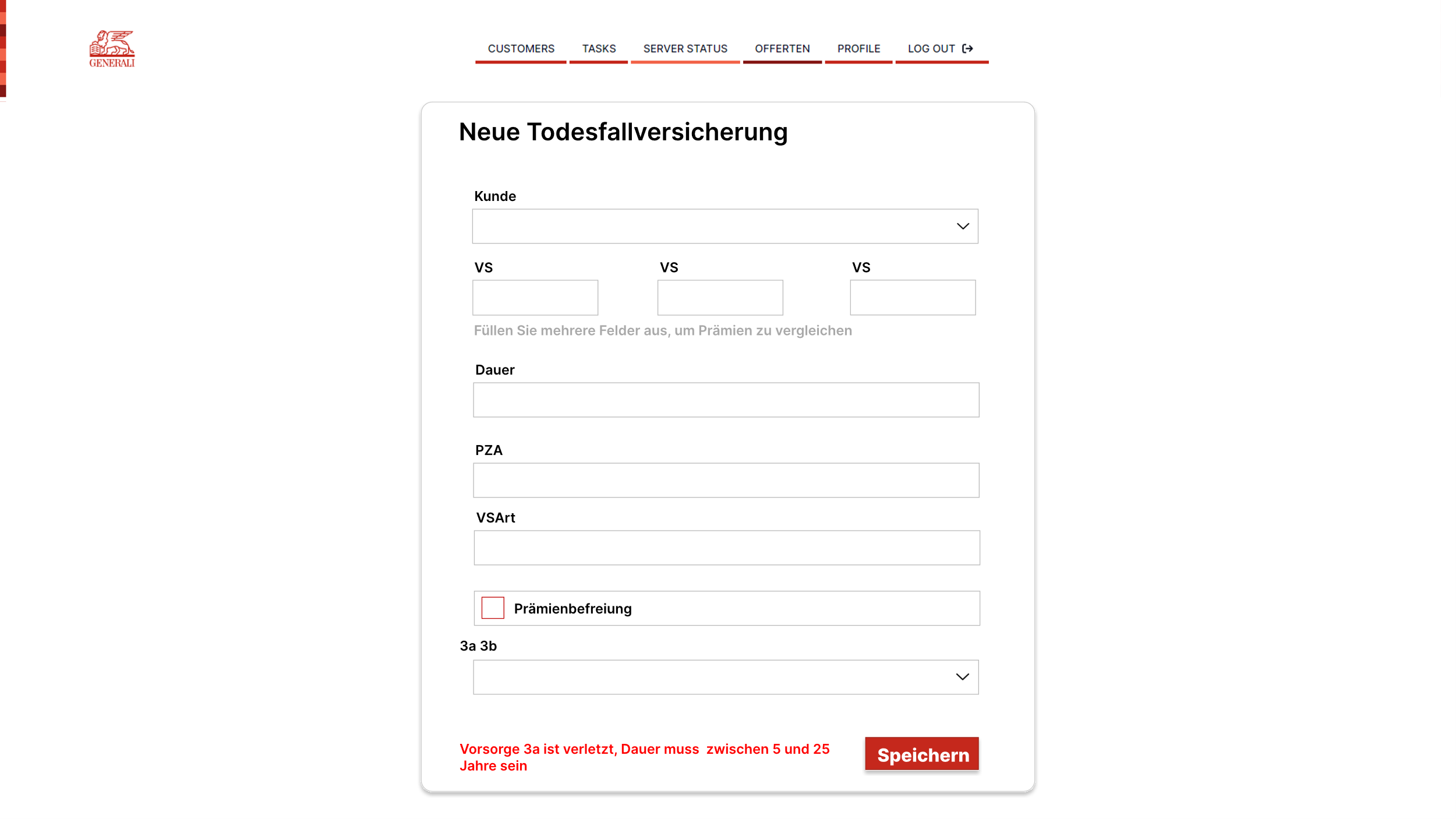
**Unittests**

Wo möglich, müssen automatisierte Unit-Tests eingesetzt werden.

### Mockups

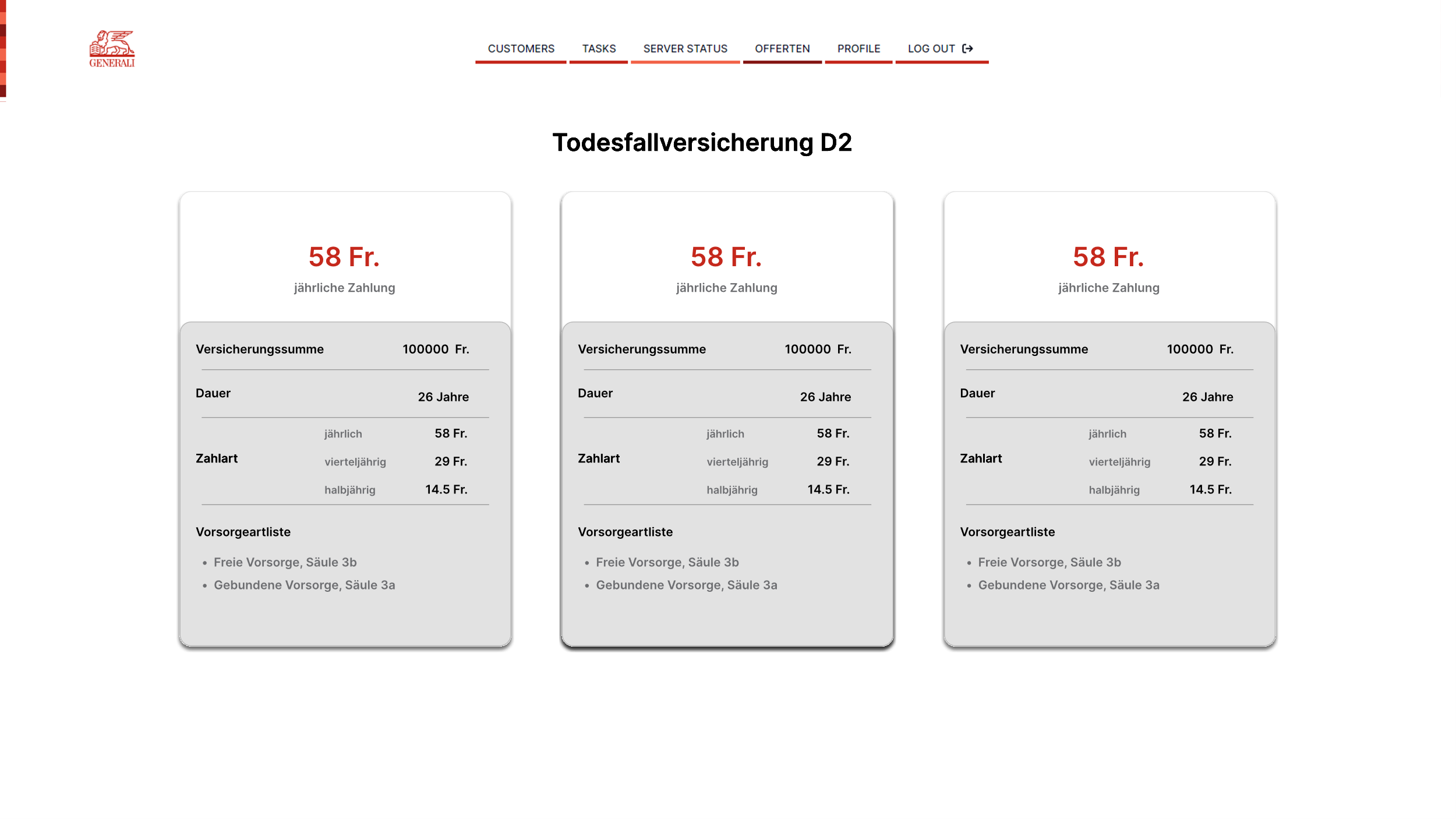
**Offer Creation**

Die Erstellungsseite wird so angepasst, dass die Eingabe mehrerer Versicherungssummen möglich ist.

**Offer Creation – Fehlermeldung**

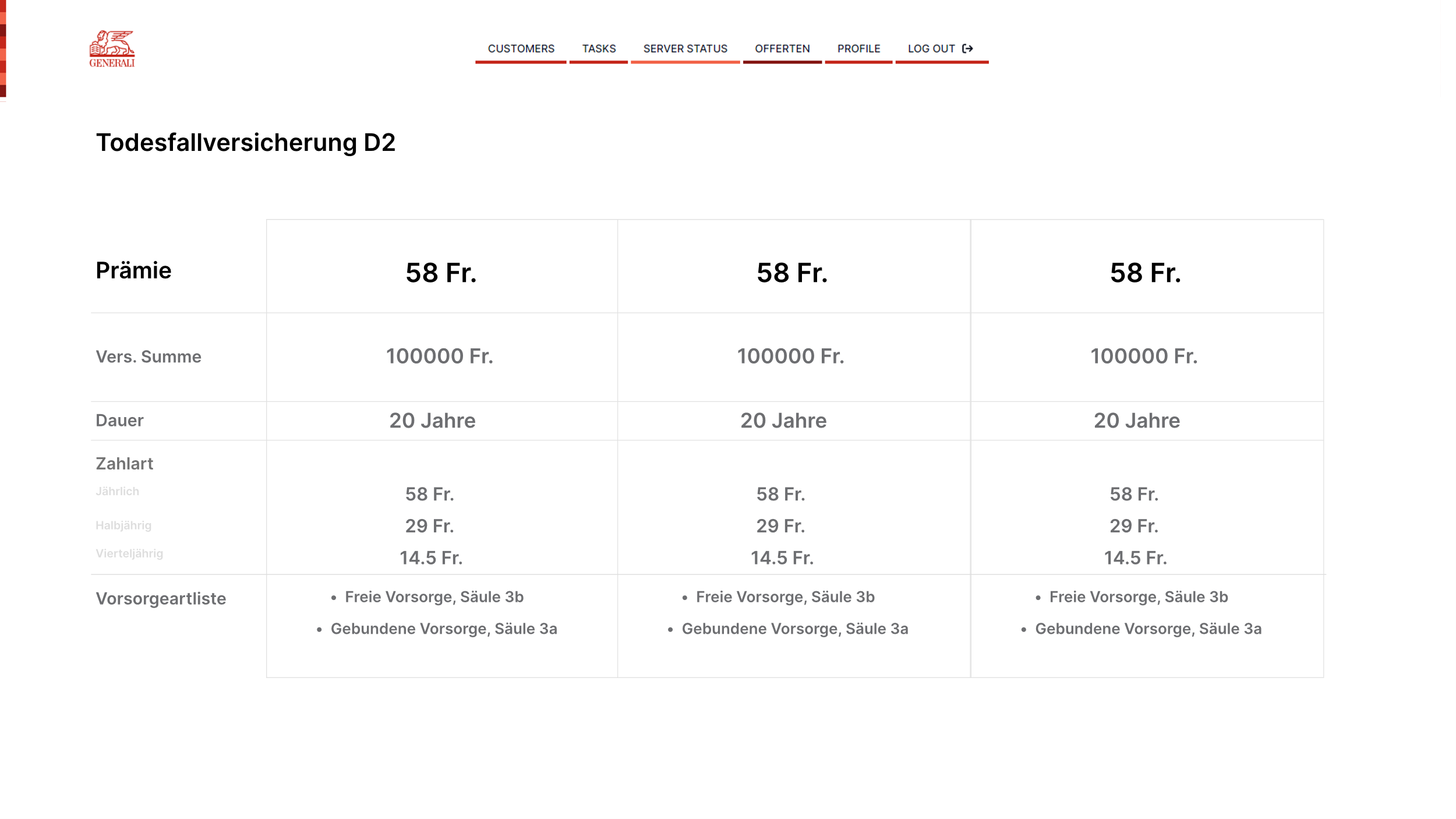
Auf der Erstellungsseite soll bei der Verletzung der Business-Rules der vom ANGEBOTSoftware generierte Fehler wie in der Abbildung dargestellt werden.

**Offer Details – Variante 1**



Das Design orientiert sich an dem der offiziellen Website von Generali, wo überwiegend Kacheln verwendet werden. Dies ermöglicht einen unmittelbaren Vergleich der Versicherungssummen.

**Offer Details – Variante 2**



Das Design ist konsistent mit dem Rest der Applikation, in der hauptsächlich Tabellen verwendet werden. Es nutzt die bereits implementierte Tabellenkomponente von Shadcn-UI.

### Testkonzept

Das Testkonzept beschreibt eine umfassende Teststrategie für dieses Projekt, welche sowohl manuelle als auch automatisierte Testverfahren beinhaltet. Die Implementierung dieser Tests findet in der Phase "Kontrollieren" nach der IPERKA-Methode statt. Dadurch wird am Ende des Entwicklungszyklus eine ganzheitliche Evaluation ermöglicht. Diese Vorgehensweise stellt sicher, dass die Projektziele erreicht und die Anforderungen gemäss der Aufgabenstellung korrekt umgesetzt wurden.

#### Testziele und Strategie

Das Ziel des Testprozesses ist, die Funktionsfähigkeit, Sicherheit, Benutzerfreundlichkeit und Zuverlässigkeit der neuen Implementierungen zu gewährleisten. Dabei wird geprüft, ob die Software die in den User Stories und Akzeptanzkriterien festgelegten Anforderungen erfüllt. Zusätzlich wird untersucht, ob sie die funktionalen so wie auch die nicht funktionalen Anforderungen und spezifischen Kriterien erfüllt sind.

Es wird ebenfalls getestet, damit Fehler und Sicherheitslücken vor der Inbetriebnahme erkannt werden können.

Um zu testen, werden, wo möglich, automatisierte Tests benutzt. Im Frontend mittels JEST und im Backend mit JUnit-Tests.   
Falls dies nicht möglich ist, werden basierend auf den User Stories und Akzeptanzkriterien manuelle Tests eingesetzt.

***Was bewusst nicht getestet wird:***

Im Testprozess werden bestimmte Bereiche bewusst ausgelassen. Load-Tests werden nicht durchgeführt, da die Benutzer hauptsächlich Berater sind und keine Endkunden. Die Zahl dieser Nutzer ist gering, was bedeutet, dass das System nicht an seine Leistungsgrenzen stösst.   
Ebenso wird die mobile Nutzererfahrung nicht getestet, da unsere Anwendung als Ersatz für eine Desktop-Anwendung dient und ausschliesslich auf Desktop-Systemen verwendet wird. Diese Entscheidungen beruhen auf der spezifischen Nutzungssituation unserer Enterprise-Software und der begrenzten Anzahl von Benutzern.

#### Testumgebung

**Gerät:** Lenovo ThinkBook 15 G4 ABA

**Betriebssystem:** Windows 11 Pro

**Systemtyp:** 64-Bit

**Prozessor:** AMD Ryzen 7 5825U with Radeon Graphics (2.00 GHz)

**RAM:** 16 GB 3200 MHz

**Browser:** Chrome Version 122.0.6261.128 (Offizieller Build) (64-Bit)

**URL:** Frontend: <http://localhost:3000>

Backend: <http://localhost:8080>

#### Testcases

|  |  |
| --- | --- |
| **Testcase 1** | |
| **Anforderung:** | Story 1, AC 1.1 – 1.2 |
| **Beschreibung** | Der Berater kann eine Offerte mit bis zu 3 Versicherungssummen erstellen. |
| **Voraussetzung** | Der Berater ist eingeloggt und hat auf der Offerten Erstellungsseite navigiert. |
| **Schritte** | Der Berater füllt das Formular aus und gibt eine bis drei Versicherungssummen an. |
| **Erwartetes Resultat** | Die Applikation zeigt drei Felder für die Versicherungssumme an. |

|  |  |
| --- | --- |
| **Testcase 2** | |
| **Anforderung:** | Story 2, AC 2.1 |
| **Beschreibung** | Offerten vergleichen |
| **Voraussetzung** | Der Berater ist eingeloggt und hat entweder eine Offerte mit zwei oder drei Versicherungssummen erstellt. |
| **Schritte** | Nach der erfolgreichen Erstellung einer neuen Offerte wird der Berater auf die Detailseite weitergeleitet. Ebenfalls kann man von der Offerten-Dashboardseite aus auf das „View Details“ Button klicken, um auf die Detailseite zu kommen. |
| **Erwartetes Resultat** | Die Offerte wird aus der Datenbank abgerufen und berechnet . Falls der Benutzer zwei oder drei Versicherungssummen angegeben hat, wird ein Bundle von berechneten Offerten zum Vergleich dargestellt. Falls nur eine Versicherungssumme eingegeben wurde, wird nur eine berechnete Offerte angezeigt. |

|  |  |
| --- | --- |
| **Testcase 3** | |
| **Anforderung:** | Story 3, AC 2.1 – 2.3 |
| **Beschreibung** | Offerten Erstellung, Validierung |
| **Voraussetzung** | Der Berater ist eingeloggt und hat auf der Offerten Erstellungsseite validiert und die Daten eingefüllt. |
| **Schritte** | Der Berater füllt die Felder mit Daten ein und versucht die Offerte mittels dem Speichern Knopf zu erstellen. |
| **Erwartetes Resultat** | Falls die Offerte die Business-Rules verletzt:   * Die Offerte wird nicht in der Datenbank abgespeichert. * Der Berater wird nicht auf die Detailseite weitergeleitet wie zuvor. * Dem Berater wird der von der ANGEBOTSoftware angegebene Fehler angezeigt. |

|  |  |
| --- | --- |
| **Testcase 4** | |
| **Anforderung:** | Story 4, AC 4.1 – 4.2 |
| **Beschreibung** | Benutzerfreundlichkeit der Detailseite |
| **Voraussetzung** | Der Berater ist eingeloggt und hat eine Offerte erfolgreich erstellt. |
| **Schritte** | Der Berater erstellt erfolgreich eine neue Offerte oder navigiert auf eine mittels der Tabelle auf der Übersichtsseite. |
| **Erwartetes Resultat** | Die Offerte und deren Daten werden klar dargestellt. Die Prämie steht im Vordergrund. Falls mehrere Versicherungssummen für die Offerte eingegeben wurden, werden die berechneten Offerten nebeneinander und einfach zum Vergleichen aufgezeichnet. |

|  |  |
| --- | --- |
| **Testcase 5** | |
| **Anforderung:** | Story 5, AC 5.1 – 5.4 |
| **Beschreibung** | Implementierung Observability-Logging |
| **Voraussetzung** | Es werden Aktionen im OVWEB System ausgeführt. |
| **Schritte** | Überprüfen der Existenz von Log-Daten. |
| **Erwartetes Resultat** | Die Logs enthalten strukturierte und relevante Informationen.  Sensible Daten werden nicht geloggt. |

|  |  |
| --- | --- |
| **Testcase 6** | |
| **Anforderung:** | Story 6, AC 6.1 – 6.3 |
| **Beschreibung** | Sicherstellung der Nachverfolgbarkeit durch Trace-ID’s in Grafana. |
| **Voraussetzung** | Trace-IDs wurden sowohl im Backend als auch im Frontend richtig und überall, wo nötig, eingesetzt.  Eine Anfrage wird ausgeführt. |
| **Schritte** | Überprüfen der Traces in Grafana Web Interface. |
| **Erwartetes Resultat** | Jede Anfrage sollte mit einer einzigartigen Trace-ID erstellt werden, die durch alle Services propagiert werden soll. In Grafana sollte ein Baum den Prozess der ausgeführten Methoden veranschaulichen. |

|  |  |
| --- | --- |
| **Testcase 7** | |
| **Anforderung:** | Story 7, AC 7.1 – 7.3 |
| **Beschreibung** | Überprüfung der automatisierten Testabdeckung und der Effektivität der Tests für die neu implementierten Funktionen, sowohl im Frontend als auch im Backend. |
| **Voraussetzung** | Neue Funktionen wurden hinzugefügt und Tests wurden geschrieben. |
| **Schritte** | Ausführung der neu erstellten Tests. |
| **Erwartetes Resultat** | Die Tesfälle decken alle neuen Funktionen ab. Unit-Tests im Backend und JEST Tests im Frontend. Die Resultate der Tests werden dokumentiert. |

|  |  |
| --- | --- |
| **Testcase 8** | |
| **Anforderung:** | Story 8, AC 8.1 – 8.2 |
| **Beschreibung** | Überprüfung der Modularität und Dokumentation der UI  Komponenten im Storybook. |
| **Voraussetzung** | Es wurden neue UI-Komponenten erstellt. |
| **Schritte** | * Inspektion der Struktur der UI-Komponenten im Storybook. * Überprüfung der Zusammensetzung komplexerer Komponenten aus einfacheren Bausteinen. * Überprüfung der Dokumentation der UI-Komponenten im Storybook auf Vollständigkeit und Verständlichkeit. |
| **Erwartetes Resultat** | * Komplexere UI-Komponenten sind modular aus einfacheren Komponenten aufgebaut. * Alle UI-Komponenten sind verständlich im Storybook dokumentiert, was ihre Wiederverwendbarkeit und Anpassbarkeit unterstützt. |

|  |  |
| --- | --- |
| **Testcase 9** | |
| **Anforderung:** | Story 9, AC 9.1 – 9.2 |
| **Beschreibung** | Überprüfung der Einbindung neuer Seiten in Figma unter Verwendung der spezifizierten Farbpalette. |
| **Voraussetzung** | Es wurden neue Seiten im Rahmen des Projekts entworfen.  Schritte: |
| **Schritte** | * Überprüfung, ob alle neuen Seiten im vorgegebenen Figma-Projekt erfasst sind. * Inspektion der Seitenentwürfe auf die Verwendung der Figma-Farbpalette. * Überprüfung der Konsistenz der Seitenentwürfe mit den Designstandards des Projekts. |
| **Erwartetes Resultat** | * Alle neuen Seiten sind korrekt im Figma-Projektfile erfasst. * Die Figma-Farbpalette wird in allen Seitenentwürfen verwendet. * Die Entwürfe erfüllen die Designstandards und tragen zur visuellen Konsistenz des Projekts bei. |

### Kriterien für die Entscheidungsphase

In diesem Kapitel werden die Kriterien für die Entscheidungsphase definiert. Es wird entschieden, welche der zwei Varianten der Offertendetailseite im Kapitel 9.2 implementiert werden sollte. Da die Benutzererfahrung und Ästhetik Teil der IPA-Kriterien sind, werden sie hier gleich gewichtet. Ebenfalls steht die Komplexität auch auf dem gleichen Niveau.

|  |  |
| --- | --- |
| **Kriterium** | **Begründung** |
| **Benutzererfahrung** | Die Benutzererfahrung steht im Vordergrund, weil es wichtig ist, dass der Prämienvergleich dem Endkunden leicht fällt. |
| **Komplexität** | Wenn die Implementation einfacher ist, bleibt mehr Zeit für andere Aufgaben. |
| **Ästhetik** | Die Ästhetik ist wichtig für eine gute Benutzerfreundlichkeit und Klarheit. |

## Entscheiden

Es wird zwischen den zwei Varianten für das Design der Detail Seite im Kapitel 9.2 entschieden.

|  |  |  |
| --- | --- | --- |
| **Kriterium** | **Variante 1** | **Variante 2** |
| Benutzererfahrung | **+** | **-** |
| Komplexität | **=** | **=** |
| Ästhetik | **+** | **-** |
| **Bewertung:** | **++** | **--** |

**Begründung der Bewertungen**

Benutzererfahrung:

In der Variante 1 steht die Prämie mehr im Vordergrund, da alle anderen Daten viel kleiner sind. In der zweiten Variante steht sie zwar auch im Vordergrund, da die anderen Daten grösser sind und die Tabelle auch ein wenig ablenkt, könnte es sein, dass man nicht zuerst auf die Prämie schaut.

Komplexität:

Beide Varianten sind relativ komplex umzusetzen. Die erste, weil es schwierig sein kann, die Elemente richtig zu positionieren, und die zweite, weil im OVWEV-Frontend Tabellen vom Komponenten Library Shadcn-UI benutzt werden, welche aus Erfahrung recht schwierig sein können, umzusetzen.

Ästhetik:

Da diese Seite dem Kunden präsentiert werden soll, wurde für Variante 1 entschieden. Diese entspricht dem Design für Endkunden auf generali.ch, wo ebenfalls Kacheln statt Tabellen eingesetzt werden.

## Realisieren

### Datenbank

Es wurden wenige Änderungen an der Datenbank vorgenommen. Wie im vorherigen Kapitel geplant, mussten zwei neue Felder in der *life\_insurance\_request*-Tabelle hinzugefügt werden. Ebenfalls wurde das bisher existierende Feld für Klarheit von *„vs“* auf *„vs1“* umbenannt. Dies wurde durch Löschen und Neuerstellung des Columns erreicht. In einem Produktionsumfeld würde dieser Ansatz vermieden, da er zum Verlust von Daten führen kann. Da die OVWEB jedoch noch in der Entwicklungsphase ist und die Datenbank lediglich mit Beispieldaten für Tests gefüllt wird, stellt dieser Ansatz hier kein Risiko dar.

Für das Update der Datenbank wurde eine Flyway Migration benutzt. Das File musste wie folgend benannt werden:

<Prefix><Version>\_\_<Description>.sql

In dem Fall wurde das File auf V1\_0\_17\_\_extend\_life\_insurance\_request\_entity.sql genannt.

Durch folgenden Code wurde die Migration dann fehlerfrei ausgeführt.

ALTER TABLE if exists "life\_insurance\_request"  
DROP COLUMN vs;  
  
ALTER TABLE if exists "life\_insurance\_request"  
 ADD COLUMN vs1 INT,  
 ADD COLUMN vs2 INT,  
 ADD COLUMN vs3 INT;

### Backend

#### Änderungen an Models, DTOs und Mappers

Aufgrund der Änderungen an der Datenbank und zur Implementierung neuer Funktionalitäten mussten existierende Models und DTOs angepasst werden. Die bestehende Attributbezeichnung „vs“ wurde zu „vs1“ geändert, und zwei neue Attribute, „vs2“ und „vs3“, wurden hinzugefügt. Zudem wurden Anpassungen an den Models und DTOs vorgenommen, um die Validierung im Frontend zu ermöglichen.

**LifeInsuranceRequestWithCustomerInfo Model**

Dieser Dto entspricht der Tabelle in der Datenbank, wo alle Daten, die für die Berechnung einer Offerte nötig sind, gespeichert sind.

Hier mussten die Attribute angepasst werden, um die neuen Versicherungssummen zu integrieren.

Weil es sein könnte, dass ein Offerten-Bundle aber nur eine Offerte enthält und somit nur eine Versicherungssumme, werden die Attribute „vs2“ und „vs3“ als **nullable = true** markiert. Die Annotation @Column verbindet die Attribute zu einem Feld in der Datenbanktabelle.

@Column(nullable = false)  
private Integer vs1;  
  
@Column(nullable = true)  
private Integer vs2;  
  
@Column(nullable = true)  
private Integer vs3;

**LifeInsuranceRequestPostDto und LifeInsuranceRequestWithCustomerInfoGetDto**

Hier mussten mit Ausnahme von den @Column Annotationen die gleichen Änderungen wie im obigen Model vorgenommen werden, um die „vs1“, „vs2“ und „vs3“ Attribute zu implementieren.

Die Annotation @NotNull beschreibt, dass die Attribute nicht leer sein dürfen.

@NotNull Integer vs1,  
Integer vs2,  
Integer vs3,

**LifeInsuranceOfferDto**

In diesem Dto mussten Änderungen vorgenommen werden, damit die Validierung im Frontend implementiert werden konnte. Das LifeInsuranceOfferDto wird benutzt, um berechnete Offerten zwischen dem Backend und Frontend der OVWEB Applikation zu übergeben, sprich auf der Detailseite.

Anpassungen mussten vorgenommen werden, damit im Frontend überprüfbar ist, ob eine Offerte erfolgreich oder nicht berechnet wurde. Dies ist Teil des Error Handling für die Berechnung der Offerten, nicht für die Validierung.

Die LifeInsuranceOfferDto beinhaltet drei wichtige Properties.

@NotNull Status status, OfferDetail offerDetail, @NotNull List<String> error) {

Der Status und der Error Code werden mit jeder Antwort der ANGEBOTSoftware zurückgewiesen. Die Offertendetails nur dann, wenn die Berechnung erfolgreich war.

Der Status beschreibt, ob die Offerte erfolgreich berechnet wurde oder nicht. Dieser kann ein SUCCESS oder ein ERROR sein.

public enum Status {  
 *SUCCESS*("0"),  
 *ERROR*("-9999");

Die offerDetails beinhaltet alle Daten einer erfolgreich berechneten Offerte. Dieses Attribut wurde im Rahmen dieser IPA neu erstellt. Zuvor befangen sich alle Daten der OfferDetails direkt im LifeInsuranceOfferDto.

Zuletzt beinhaltet Error die von der ANGEBOTSoftware generierte Fehler, falls Business-Rules verletzt wurden.

Die Offertendetails wurden nicht mit der @NotNull Annotation markiert, da es sein kann, dass die Offerte nicht erfolgreich berechnet wird, in welchem Fall die OfferDetails tatsächlich null wären.  
Falls die Offertendetails jedoch vorhanden sind, wurden alle darin beinhaltete Attribute mit @NotNull markiert, um sicherzustellen, dass auch alles richtig übergeben wird.

**DtoMapper**

Weil sich die Struktur des LifeInsuranceOfferDtos wie oben beschrieben geändert hat, mussten Änderungen am Mapper vorgenommen werden.

Es wurde eine neue Variable offerDetails von Typ LifeInsuranceOfferDto.OfferDetail programmiert. In der werden die OffertenDetails folgend eingefügt.

var offerDetail =  
 new LifeInsuranceOfferDto.OfferDetail(  
 calculation.getOverview(),  
 berechnungsliste,  
 calculation.getGraphScaleMaxValue(),  
 praemienzahlartliste,  
 vorsorgeartliste,  
 avbListe);

#### Controller und Services

Um diese Änderungen an den Models, DTOs und Mappers einzusetzen, mussten die Services und Controller geändert werden.

Wie im Kapitel 8.3.3 beschrieben, sind die Controllers und Service wie folgt benannt:

LifeInsuranceRequest –> Verwaltung der Offerten

LifeInsuranceCalculation -> Berechnung der Offerten

Für alle hier beschriebenen Methoden wurde die Annotation @NewSpan hinzugefügt, um Tracing mittels Grafana zu ermöglichen.

Für die mapToFosoftRequest und addCustomerInfoToRequest Methoden wurde dies bewusst nicht getan, weil nur Spring-Bean Methoden mit @NewSpan annotiert werden können.

**LifeInsuranceCalculationController**

@GetMapping("/{id}")  
 public List<LifeInsuranceOfferDto> getOfferDetails(@PathVariable Long id) {  
 *log*.atInfo()  
 .setMessage("Getting offer details for request ID:" + id)  
 .addKeyValue("requestId", id)  
 .log();  
  
 List<ProductCalculationResponse> responses = calculatorService.calculateOffer(id);  
 List<LifeInsuranceOfferDto> results =  
 responses.stream().map(DtoMapper.*INSTANCE*::mapToDto).collect(Collectors.*toList*());  
 return results;  
 }  
}

Der Controller wird angepasst, um mehrere Berechnungen zurückzugeben. Anstatt nur eine LifeInsuranceOfferDto wird eine Liste von LifeInsuranceOfferDtos zurückgegeben. Ein Log wird herausgegeben, das beschreibt, welche Offerte berechnet werden soll.

**LifeInsuranceRequestController.java**

**Methode:** *createLifeInsuranceRequest*

In der LifeInsruanceRequestController, spezifisch in der createLifeInsuranceRequest Methode, wird die Validierung sowie das Verwalten von bis zu drei Offerten implementiert.

Um dies zu ermöglichen, wurde das LifeInsuranceCalculationService in der Klasse initialisiert und zum Constructor hinzugefügt.

Im LifeInsuranceCalculationService wird dann die tatsächliche Logik der neuen checkOfferValidity Methode implementiert. Hier wird sie aufgerufen und basiert auf den Antworten gehandelt.

List<ProductCalculationResponse> validityResponses =  
 calculatorService.checkOfferValidity(  
 request.customerId(), DtoMapper.*INSTANCE*.mapToEntity(request));

Der Methode checkOfferValidity wird der Request zusammen mit der customerId übergeben. Anschliessend wird der Request entsprechend der erforderlichen Struktur gemappt, in diesem Fall zu einem LifeInsuranceRequestWithCustomerInfo-Modell.

Die Antwort ist eine Liste von ProductCalculationResponses.

Nachdem die Antwort erhalten wird, wird über die Liste iteriert und die Variable Status wird verifiziert.

for (ProductCalculationResponse response : validityResponses) {  
 if (response.getResult().getFirst().getStatus() != Status.*\_0*) {  
 throw new ResponseStatusException(  
 HttpStatus.*BAD\_REQUEST*,  
 response.getResult().getFirst().getError().getFirst().getValue());  
 }  
}

Falls diese in jener der berechneten Offerten nicht 0 ist, wird ein Error Code so wie der vom ANGEBOTSoftware generierte Fehler zurückgeben, damit die Verletzung der Business-Rules auch im Frontend dargestellt werden kann.

Falls alle Berechnungen erfolgreich sind, ist die Validierung erfolgreich. Die Daten können mittels LifeInsuranceRequestService in der Datenbank gespeichert werden.

var info =  
 service.createLifeInsuranceRequest(  
 keyCloakId, request.customerId(), DtoMapper.*INSTANCE*.mapToEntity(request));  
return DtoMapper.*INSTANCE*.mapToGetDto(info);

In dieser Methode wird geloggt, für welchen Keycloak User die Methode validiert wird.

*log*.atInfo()  
 .setMessage("Creating insurance request for userKeyCloakId:" + keyCloakId)  
 .addKeyValue("userKeycloakId", keyCloakId)  
 .log();

**LifeInsuranceCalculationService**

Im LifeInsuranceCalculationService wurden die grössten Änderungen am Backend vorgenommen. Hier wurde die Logik zum Berechnen und zum Validieren einer Offerte vorgenommen.

**Methode:** *checkOfferValidity*

Diese Methode wird für die Validierung der Offerten verantwortlich sein.

Zuerst wird geloggt, dass die Offerte validiert wird und für welchen Kunden ID dies getan wird

*log*.atInfo()  
 .setMessage("Validating insurance request for customer with id " + customerId)  
 .addKeyValue("customerId", customerId)  
 .log();

Weil die Customer ID aus dem Controller mitgeben wird und die Kundendaten für die Validierung notwendig sind, werden diese zuerst aus der Datenbank abgerufen. Dies wird mittels Customerrepositories gemacht, welches importiert und zum Constructor hinzugefügt wurde.

Customer customer =  
 customerRepository  
 .findById(customerId)  
 .orElseThrow(  
 () -> new EntityNotFoundException("Customer not found with ID: " + customerId));

Dies gibt die Kundendaten zurück.

Da nun möglicherweise ein Offerten-Bundle validiert werden muss, das mehrere Versicherungssummen beinhaltet und somit mehrere Berechnungen erfordert, werden die Versicherungssummen aus dem Request extrahiert und in einer Liste gespeichert.

Integer[] vsValues = {request.getVs1(), request.getVs2(), request.getVs3()};

Mithilfe dieser Liste kann nun über jede Versicherungssumme iteriert und somit für jede eine Validierung durchgeführt werden. Die Validierung erfolgt mittels der Methode calculateLifeInsurance, die entweder mit einer berechneten Offerte oder einem Fehler antwortet. Jede dieser Responses wird in einer Liste vom Typ ProductCalculationResponse gespeichert.

for (Integer vs : vsValues) {  
 if (vs != null) {  
 try {  
 ProductCalculationRequest foSoftRequest =  
 LifeInsuranceCalculationService.*mapToFoSoftRequest*(  
 *addCustomerInfoToRequest*(request, customer), vs);  
 ProductCalculationResponse response =  
 foSoftClient  
 .calculateLifeInsurance(Language.*DE*, "01.02.2024", foSoftRequest)  
 .getBody();

*log*.atInfo()  
 .setMessage("Validating offer with vs value :" + vs)  
 .addKeyValue("vs", vs)  
 .log();

Im for-Schleife wird zunächst geprüft, ob die Versicherungssumme null ist. Diese Überprüfung ist erforderlich, da ein Berater eventuell ein Offerten-Bundle mit nur zwei Offerten erstellt hat. In einem solchen Fall wäre die dritte Versicherungssumme null, und es sollte folglich keine Validierung oder Berechnung stattfinden. Anschliessend wird ein fosoftRequest erstellt. Die vorhandenen Daten werden so angepasst, dass sie der erforderlichen Struktur für den Endpoint D2 der ANGEBOTSoftware entsprechen, wie in Kapitel 8.3.3 beschrieben. Zudem wird die Methode addCustomerInfoToRequest neu implementiert, welche notwendig ist, um die Kundendaten auf den fosoftRequest zu mappen.

Nachdem der fosoftRequest erstellt wurde, wird dieser mittels der bestehenden calculateLifeInsurance Methode verwendet, um die Offerte zu berechnen. Diese wird dann in Response gespeichert.

Da diese Antwort bei der Verletzung der Business-Rules auch einen Fehler beinhalten kann, wird im folgenden Code geprüft, ob die Antwort ein Fehler ist.

Es wird hierbei auch geloggt, welche Versicherungssumme validiert wird.

if (response == null || response.getResult().get(0).getStatus() != Status.*\_0*) {  
 Error error = new Error();  
 String errorMessage = "VS Summe: " + vs + ":";  
 errorMessage += " Error: " +

response.getResult().get(0).getError().get(0).getValue();  
 error.setValue(errorMessage);  
 ProductCalculation errorCalculation = new ProductCalculation();  
 errorCalculation.addErrorItem(error);  
 ProductCalculationResponse errorResponse = new ProductCalculationResponse();  
 errorResponse.addResultItem(errorCalculation);  
 responses.add(errorResponse);  
} else {  
 responses.add(response);  
}

Die Antwort Status = 0 würde darauf hindeuten, dass die Offerte erfolgreich berechnet worden ist. Falls das nicht der Fall ist, wird der Fehler aus der Response herausgeholt und mittels eines neuen ProductCalculation Objekts zu der Liste der Responses hinzugefügt.

Dies wird gemacht, damit dem Frontend auch übergeben wird, bei der Berechnung welcher Versicherungssumme das Fehler geschehen ist.

Im Falle eines Fehlers mit dem API-Call wird ein Catch mit Logging implementiert.

catch (Exception e) {  
 *log*.atInfo()  
 .setMessage("Validation failed for vs with sum: " + vs)  
 .addKeyValue("vs", vs)  
 .log();  
 Error error = new Error();  
 error.setValue("Unexpected error for VS value: " + vs + ". Error: " + e.getMessage());  
 ProductCalculation errorCalculation = new ProductCalculation();  
 errorCalculation.addErrorItem(error);  
 ProductCalculationResponse errorResponse = new ProductCalculationResponse();  
 errorResponse.addResultItem(errorCalculation);  
 responses.add(errorResponse);  
}

Hier wird aber ein anderer Fehler am Frontend übergeben, damit klar erkannt werden kann, dass nicht eine Verletzung der Business-Rules dargestellt ist, sondern ein Fehler mit dem API-Call.

**Methode:** *addCustomerInfoToRequst*

Diese Methode wurde beim Refactoring des Codes implementiert. Gemerkt wurde, dass die neu geschriebene Methode, mapToFoSoftRequestValidityCheck, redundant war, wie hier zu sehen ist:

static ProductCalculationRequest mapToFoSoftRequestValidityCheck(  
 Customer customer, LifeInsuranceRequestWithCustomerInfo request, Integer vs) {  
 var targetFormat = DateTimeFormatter.*ofPattern*("dd.MM.yyyy");  
 var formattedDate = customer.getBirthdate().format(targetFormat);  
  
 var requestPayload = new ProductCalculationRequest();  
 var personData = new ProductCalculationRequestPersonendaten();  
 var insuranceData = new ProductCalculationRequestBerechnungslisteInner();  
 personData.setGeburtsdatum(formattedDate);  
 personData.setNationalitaet("CH");  
 personData.setRaucher(customer.getSmoker());  
 insuranceData.setDauer(request.getPolicyPeriod());  
 insuranceData.setVS(vs);  
 insuranceData.setPZA(PremiumInstallmentsYear.*fromValue*(request.getPza().getValue()));  
 insuranceData.set3a3b(PensionPillar.*fromValue*(request.getPensionPillar().getValue()));  
 insuranceData.setVsArt(request.getVsArt());  
 insuranceData.setPraemienbefreiung(request.getPremiumWaiver());  
 requestPayload.setPersonendaten(personData);  
 requestPayload.setBerechnungsliste(List.*of*(insuranceData));  
 return requestPayload;  
}

static ProductCalculationRequest mapToFoSoftRequest(  
 LifeInsuranceRequestWithCustomerInfo request, Integer vs) {  
 var targetFormat = DateTimeFormatter.*ofPattern*("dd.MM.yyyy");  
 var formattedDate = request.getBirthdate().format(targetFormat);  
 var gender = request.getGender().equals(Gender.*MALE*) ? 1 : 2;  
  
 var requestPayload = new ProductCalculationRequest();  
 var personData = new ProductCalculationRequestPersonendaten();  
 personData.setGeburtsdatum(formattedDate);  
 personData.setGeschlecht(  
 ProductCalculationRequestPersonendaten.GeschlechtEnum.*fromValue*(gender));  
 personData.setRaucher(request.getSmoker());  
 personData.setPLZ(request.getPlz());  
 personData.setWohnort(request.getCity());  
 personData.setNationalitaet("CH");  
  
 var insuranceData = new ProductCalculationRequestBerechnungslisteInner();  
 insuranceData.setDauer(request.getPolicyPeriod());  
 insuranceData.setVS(vs);  
 insuranceData.setPZA(PremiumInstallmentsYear.*fromValue*(request.getPza().getValue()));  
 insuranceData.set3a3b(PensionPillar.*fromValue*(request.getPensionPillar().getValue()));  
 insuranceData.setVsArt(request.getVsArt());  
 insuranceData.setPraemienbefreiung(request.getPremiumWaiver());  
  
 requestPayload.setPersonendaten(personData);  
 requestPayload.setBerechnungsliste(List.*of*(insuranceData));  
 return requestPayload;  
}

Im Code wird sichtbar, dass die Methoden das Gleiche tun, nur sind die Parameter leicht anders. Die Methode addCustomerInfoToRequest fügt die Kundendaten zum LifeInsuranceRequestWithCustomerInfo, bevor Sie der mapToFosoftRequest Methode übergeben wird. Somit kann die redundante Methode wieder gelöscht werden.

static LifeInsuranceRequestWithCustomerInfo addCustomerInfoToRequest(  
 LifeInsuranceRequestWithCustomerInfo request, Customer customer) {  
 request.setBirthdate(customer.getBirthdate());  
 request.setGender(customer.getGender());  
 request.setSmoker(customer.getSmoker());  
 request.setPlz(customer.getAddress().getPlz());  
 request.setCity(customer.getAddress().getCity());  
 return request;  
}

**Methode:** *calculateOffer*

Diese Methode muss auch angepasst werden, um bis zu drei Offerten zu handhaben. Dies ist aus dem Grund, weil sie auch direkt via des Endpoints /calculateOffer/{id} im entsprechenden Controller von der Detailseite aus aufgerufen wird.

@Operation(summary = "Create a LifeInsuranceRequest")  
@PostMapping  
public LifeInsuranceRequestWithCustomerInfoGetDto createLifeInsuranceRequest(  
 @RequestBody LifeInsuranceRequestPostDto request, @AuthenticationPrincipal Jwt principal) {  
 var keyCloakId = principal.getSubject();  
 *log*.atInfo()  
 .setMessage("Creating insurance request for userKeyCloakId:" + keyCloakId)  
 .addKeyValue("userKeycloakId", keyCloakId)  
 .log();  
  
 List<ProductCalculationResponse> validityResponses =  
 calculatorService.checkOfferValidity(  
 request.customerId(), DtoMapper.*INSTANCE*.mapToEntity(request));  
  
 for (ProductCalculationResponse response : validityResponses) {  
 if (response.getResult().getFirst().getStatus() != Status.*\_0*) {  
 throw new ResponseStatusException(  
 HttpStatus.*BAD\_REQUEST*,  
 response.getResult().getFirst().getError().getFirst().getValue());  
 }  
 }  
 var info =  
 service.createLifeInsuranceRequest(  
 keyCloakId, request.customerId(), DtoMapper.*INSTANCE*.mapToEntity(request));  
 return DtoMapper.*INSTANCE*.mapToGetDto(info);  
}

Hier wird ähnliche Logik wie in der checkOfferValidity Methode implementiert. Es wird über die Versicherungssummen iteriert und die Offerten werden berechnet.

Der Unterschied liegt hier darin, dass das Frontend nur die ID der Offerte mitgibt. Damit werden die Offertendaten mittels LifeInsuranceRequestRepositroies aus der Datenbank ausgelesen.

### Frontend

#### Actions

Um das Erlauben der Erstellung eines Offerten-Bundles so wie dessen Validierung zu ermöglichen, müssen Änderungen an den Server Actions vorgenommen werden.

*create-offer*

'use server'  
import {  
 CreateLifeInsuranceRequestRequest,  
 LifeInsuranceRequestControllerApi,  
 ResponseError  
} from '@it-apprentices/ovweb'  
import {withApi} from '@/lib/with-api'  
import {withSpan} from '@/lib/with-span'  
import {getLogger} from '@/logging/log-util'  
import {nonNullish} from '@/types/guards'  
  
export const createOffer = async (  
 request: CreateLifeInsuranceRequestRequest  
) => {  
 return await withSpan('createOffer', {}, async span => {  
 const logger = getLogger('ovweb-frontend')  
 const logContext: Record<string, unknown> = {}  
 logger.info('creating Offer')  
 try {  
 return await withApi(  
 LifeInsuranceRequestControllerApi  
 ).createLifeInsuranceRequest(request)  
 } catch (error) {  
 const response = (error as ResponseError | undefined)?.response  
 const responseBody = await response  
 ?.text()  
 .catch(() => 'failed to read response body')  
  
 if (nonNullish(response)) {  
 const responseStatus = response.status  
 const responseStatusText = response.statusText  
 span.setAttribute('responseStatus', responseStatus)  
 span.setAttribute('responseStatusText', responseStatusText)  
 span.setAttribute(  
 'responseBody',  
 responseBody ?? 'no response body'  
 )  
 logContext.responseStatus = responseStatus  
 logContext.responseStatusText = responseStatusText  
 logContext.responseBody = responseBody  
 }  
 logger.error(logContext, 'creating task failed')  
 throw new Error(responseBody)  
 }  
 })  
}

Dank der Benutzung von OpenAPI, wie im Kapitel 8.3.2 beschrieben, wurde das „CreateLifeInsuranceRequestRequest“ so wie „LifeInsuranceRequestControllerApi“ den Backend-Änderungen angepasst. Somit sind drei Versicherungssummen und die Änderungen an den Methoden vorhanden.

Die Erweiterung der Methode implementiert neues Logging. Ebenfalls wird mittels der Linie „withSpan“ Tracing zur Action hinzugefügt.

Das Fehlerhandling beinhaltet zwei Varianten. Eine, wo die Struktur der Response nicht stimmt und eine, in der die Validierung fehlgeschlagen hat.

*get-offer-details*

'use server'  
import {  
 LifeInsuranceCalculationControllerApi,  
 ResponseError  
} from '@it-apprentices/ovweb'  
import {withApi} from '@/lib/with-api'  
import {withSpan} from '@/lib/with-span'  
import {getLogger} from '@/logging/log-util'  
import {nonNullish} from '@/types/guards'  
  
export const getOfferDetails = async (id: number) => {  
 return await withSpan('getOfferDetails', {requestId: id}, async span => {  
 const logger = getLogger('ovweb-frontend')  
 const logContext: Record<string, unknown> = {requestId: id}  
 logger.info(logContext, 'getting offer details')  
 try {  
 return await withApi(  
 LifeInsuranceCalculationControllerApi  
 ).getOfferDetails({id})  
 } catch (error) {  
 const response = (error as ResponseError).response  
 if (nonNullish(response)) {  
 const responseStatus = response.status  
 const responseStatusText = response.statusText  
 const responseBody = await response  
 .text()  
 .catch(() => 'failed to read response body')  
 span.setAttribute('responseStatus', responseStatus)  
 span.setAttribute('responseStatusText', responseStatusText)  
 span.setAttribute('responseBody', responseBody)  
 logContext.responseStatus = responseStatus  
 logContext.responseStatusText = responseStatusText  
 logContext.responseBody = responseBody  
 }  
 logger.error(logContext, 'getting offer details failed')  
 throw error  
 } })

Die Änderungen in „get-offer-details“ ähneln sich denen an der „create-offer“ Action. Hier wird auch Tracing durch „withSpan“ implementiert, wobei hier die ID der angefragten Offerte auch im Trace mitgegeben wird.

Logging wird hier auch eingesetzt. In „get-offer-details“ wird die ID der Offerte auch mitgeloggt.

#### ProductCalculationDisplay Component

Für die Anzeige und den Vergleich mehrerer Offerten wird eine neue Komponente entwickelt:

const ProductCalculationDisplay: *React*.FC<ProductCalculationDisplayProps> = ({  
 praemie,  
 pzaDescription,  
 policyPeriod,  
 vs,  
 praemienzahlartliste,  
 vorsorgeartliste  
})

Damit die Komponente für verschiedene Arten von Versicherungen wiederverwendbar ist, werden diese Properties genutzt, um die Anpassungsfähigkeit und Flexibilität der Komponenten zu gewährleisten.

Das Styling der Komponenten entspricht dem Design in Figma und sieht wie folgt aus:

A screenshot of a phone

Description automatically generated

#### Seiten

A screenshot of a computer

Description automatically generated*Erstellungs Seite*

Auf der Erstellungsseite werden neue Felder zur Erfassung von bis zu drei Versicherungssummen implementiert. Zusätzlich wird eine Validierung mittels OVWEB Backend und ANGEBOTSoftware eingeführt.

Zunächst muss das ursprüngliche Feld von „VS“ in „VS1“ umbenannt werden. Diese Umbenennung erfordert Anpassungen in der Zod-Validation, im onSubmit-Handler sowie im Formular selbst. Die Logik von „VS1“ wird kopiert und zweimal dupliziert, wodurch die neuen Felder „VS2“ und „VS3“ entstehen. Es sind geringfügige Anpassungen im Styling notwendig, um sicherzustellen, dass jedes Feld auf gleicher Höhe steht und etwa ein Drittel des Parent-Divs einnimmt.

Zur Verbesserung der Benutzerfreundlichkeit werden in der useEffect-Hook Zustände verwendet, sodass das „VS2“-Feld erst benutzbar wird, wenn eine Summe in „VS1“ eingetragen wurde. Das „VS3“-Feld wird zugänglich, sobald das „VS2“-Feld befüllt ist.

Für das Empfangen und Anzeigen von Fehlern bei der Erstellung, wie die Verletzung der Business-Rules, wird der Fehler aus der entsprechenden im onSubmit ausgeführten Action benutzt:

} catch (e) {  
 *console*.error('Error while creating offer request:', e)  
 form.setError('root.formValidation', {  
 message: *JSON*.parse((e as Error).message).message  
 })

{nonNullish(form.formState.errors.root) && (  
 <div className="text-red-500 mb-4">  
 {  
 form.formState.errors.root  
 .formValidation.*message* }  
 </div>  
)}

A screenshot of a computer

Description automatically generated*Detail Seite*

Die Detailseite ermöglicht das Vergleichen von bis zu drei Offerten. Sie wird nach dem Figma Design aufgebaut.

const OfferDetails: FC<PageProps> = ({params}) => {  
 const [offerData, setOfferData] = useState<LifeInsuranceOfferDto[] | null>(  
 null  
 )

Zuerst wird die Änderung vorgenommen, dass eine Liste von LifeInsuranceOfferDtos erwartet werden soll und nicht mehr ein einziges LifeInsuranceOfferDto.

Die Offerten werden mittels useAsyncEffect asynchron mit der getOfferDetails Action abgerufen.

Sobald die Antwort der OVWEB-Backend-Applikation empfangen wird, werden die Offerten dargestellt. Um dies zu tun, wird über die vorher definierte Liste von LifeInsuranceOfferDtos iteriert, und die nötigen Attribute für den ProductCalculationDisplay werden gesetzt.

<div className="flex justify-around w-full">  
 {offers.map((calculation, index) => {  
 const {praemie, policyPeriod, vs} =  
 calculation.calculation  
 return (  
 <ProductCalculationDisplay  
 key={index}  
 praemie={praemie}  
 pzaDescription={calculation.calculation.pza}  
 policyPeriod={policyPeriod}  
 vs={vs}  
 praemienzahlartliste={  
 calculation.praemienzahlartliste  
 }  
 vorsorgeartliste={calculation.vorsorgeartliste}  
 />  
 )  
 })}  
</div>

### Automatisiertes Testing

#### Backend Unit Tests

Im Backend wurden neue Methoden zur Berechnung und Validierung von Lebensversicherungsangeboten in den LifeInsuranceCalculationService integriert. Diese Methoden wurden durch umfassende Unit Tests abgesichert, um die Funktionalität und Robustheit der Implementierung zu gewährleisten. Die bestehenden Unit Tests wurden angepasst, um das neue Attribut „vs1“ zu integrieren.

**Hinzufügen von Kundeninformationen zu Anfragen**

In der Methode addCustomerInfoToRequest werden zusätzliche Kundeninformationen zu einer bestehenden Anfrage hinzugefügt. Diese Methode ist besonders wichtig, da sie sicherstellt, dass alle relevanten Kundendetails für die Angebotsberechnung berücksichtigt werden. Der Test überprüft, ob alle Kundeninformationen korrekt in die Anfrage integriert werden.

**Validierung der Angebotsvalidität**

Die Methoden checkOfferValidity\_Successful und checkOfferValidity\_CustomerNotFound überprüfen die Validität eines Angebots basierend auf der Kunden-ID. Der erste Test bestätigt, dass ein gültiges Angebot zurückgegeben wird, wenn der Kunde in der Datenbank existiert. Der zweite Test überprüft die Fehlerbehandlung, wenn der Kunde nicht gefunden wird, und stellt sicher, dass eine angemessene Ausnahme ausgelöst wird.

Diese Tests tragen dazu bei, die Zuverlässigkeit und Genauigkeit des Versicherungsberechnungsdienstes zu gewährleisten und Fehler in der Anwendungslogik frühzeitig zu erkennen.

#### Frontend Tests

Im Frontend wurde die neue Komponente „ProductCalculationDisplay“ getestet.

Es wurden alle neuen Komponenten im Storybook des Projektes erfasst, dabei wurden positive so wie negative Szenarien integriert, wie in folgenden Beispielen.

A screenshot of a phone

Description automatically generatedA screenshot of a computer

Description automatically generated

Für folgende Komponenten wurden neue Storybook-Einträge geschrieben:

*OfferDetail Page:*

* Loading State
* Calculation Success State
* Calculation Success With 2 VS State
* Calculation Success With 3 VS State
* Calculation Business Error State
* Data Fetch Error State

*CreateInsuranceForm Component:*

* Filled State
* Filled With 2 VS State
* Filled With 3 VS State
* Empty State
* Failed Validation State
* Failed Creation State

*ProductCalculationDisplayProp Component:*

* Filled State

## Kontrollieren

### Manuelles Testing

Mittels dem zuvor erstellten Testkonzept aus dem Kapitel 9.3 werden die Tests durchgeführt.

#### Testprotokoll

Die Testumgebung für das folgende Testprotokoll kann im Kapitel 9.3.2 nachgeschlagen werden.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Testfall** | **Resultat** | **Datum** | **Tester** | **Bemerkung** |
| **1** | Passed | 15.04.2024 | Andrei Mititelu | Die Seite zeigt drei validierte Felder für die Versicherungssumme an. |
| **2** | Passed | 15.04.2024 | Andrei Mititelu | Die Seite zeigt die Offertendetails an. Im Falle eines Offerten-Bundles werden alle drei Offerten angezeigt. |
| **3** | Passed | 15.04.2024 | Andrei Mititelu | Die Validierung funktioniert wie gewollt. |
| **4** | Passed | 15.04.2024 | Andrei Mititelu | Die Offerten sind klar zum Vergleich dargestellt. |
| **5** | Passed | 15.04.2024 | Andrei Mititelu | Logging funktioniert korrekt. |
| **6** | Passed | 15.04.2024 | Andrei Mititelu | Tracing funktioniert korrekt |
| **7** | Passed | 15.04.2024 | Andrei Mititelu | Testfälle decken alle neue Funktionen ab. |
| **8** | Passed | 15.04.2024 | Andrei Mititelu | Komponente sind modular aufgebaut und im Storybook dokumentiert. |
| **9** | Passed | 15.04.2024 | Andrei Mititelu | Alle Seiten sind in Figma erfasst. |

#### Testbericht massnahmen schreiben

Gemäss dem Testprotokoll wurden alle manuellen Tests erfolgreich absolviert. Die Funktionen der Applikation wurden planmässig getestet und haben in allen Testfällen wie erwartet reagiert. Im Folgenden wird ein Fazit zu jedem einzelnen Testfall präsentiert.

**Testfall 1**

Das Form reagiert wie erwartet und zeigt drei Felder für die Versicherungssummen an. Die Versicherungssumme 2 ist erst ausfüllbar, sobald die erste ausgefüllt wurde. Die dritte, erst nachdem die zweite ausgefüllt wurde.

**Testfall 2**

Die Seite zeigt die Offertendetails an. Falls mehr als eine Versicherungssumme eingegeben wurde, werden Offerten zum Vergleich nebeneinander dargestellt.

*Massnahme / Empfehlung:*

Um die Benutzererfahrung weiter zu verbessern, könnte eine dynamische Anzeige implementiert werden, die den Beratern ermöglicht, Offerten nach Grösse der Versicherungssummen zu ordnen und Offerten auszublenden. Dies würde eine noch flexiblere Vergleichsmöglichkeit bieten.

**Testfall 3**

Die Validierung funktioniert wie erwartet. Im Falle, dass die eingegebenen Daten die Business-Rules verletzen, wird die Offerte nicht in der Datenbank gespeichert. Es wird auch nicht auf der Detailseite weitergeleitet und die Applikation zeigt dem Berater die vom ANGEBOTSoftware retournierte Fehler.

**Testfall 4**

Die Offerten sind nach Figma Design schön und übersichtlich dargestellt. Die Versicherungssumme steht im Vordergrund, was den Vergleich einfach macht. Falls mehrere Summen eingegeben wurden, werden die Offerten als Offerten-Bundle nebeneinander auf der gleichen Höhe zum Vergleich dargestellt.

**Testfall 5**

Das Logging funktioniert im Testfall 5 korrekt und liefert die notwendigen Informationen zur Überwachung und Fehleranalyse, ohne sensible Daten auszuloggen.

**Testfall 6**

Das Tracing ist erfolgreich implementiert und funktioniert wie erwartet. Dies kann man in Grafana nachverfolgen.

A screenshot of a computer

Description automatically generated

**Testfall 7**

Testfälle wurden implementiert und decken die neuen Funktionen ab. Die Resultate sind im Kapitel 12.2 dokumentiert.

*Massnahme / Empfehlung:*

Kontinuierliche Überprüfung und Erweiterung der Testfälle, um sicherzustellen, dass zukünftige Updates oder neue Funktionen vollständig abgedeckt sind und die Applikation weiterhin den Anforderungen entspricht.

**Testfall 8**

UI-Komponenten sind modular aufgebaut und wiederverwendbar. Diese wurden wie im Kapitel 11.4.2 im Storybook des Projektes dokumentiert.

**Testfall 9**

Alle Seiten sind im Figma erfasst und benutzen die Figma-Farbpalette, die dem Generali Styleguide entspricht. Die Entwürfe wurden vor der Realisierungsphase erstellt und sind konsistent mit dem Generali-Design.

Link zu Figma : <https://www.figma.com/file/AafXJNFsajpnfRFDKIhjE3/Mockups-(FoSoft-Web)?type=design&node-id=0%3A1&mode=design&t=cWQtE4kV41e038AC-1>

*Massnahme / Empfehlung:*

Periodische Überprüfung der Design-Konsistenz mit dem Generali Styleguide, besonders nach grösseren Updates oder Design-Änderungen, um eine durchgängige Markenkohärenz sicherzustellen.

### Unit und Jest Tests

Im Backend als auch im Frontend wurden Unit- und Jest-Tests durchgeführt, die alle gestellten Anforderungen erfolgreich erfüllt haben. Jeder Test hat das Ergebnis „passed“ erzielt, was bestätigt, dass sie ohne Probleme durchlaufen wurden. Diese Tests sind von entscheidender Bedeutung, um die Funktionalität der verschiedenen Komponenten und Module der Anwendung zu validieren. Das durchweg positive Testergebnis garantiert die Zuverlässigkeit und Stabilität der Anwendung. Zudem geben diese Ergebnisse die Sicherheit, dass alle notwendigen Anforderungen und Kriterien erfüllt wurden.

Folgend die Beweise für die erfolgreich durchgeführten Tests.

***Frontend:***

A screen shot of a computer

Description automatically generated

Alle im Projekt existenten Tests inklusive der neuen Implementierungen sind erfolgreich ausgeführt worden.

***Backend:***

***A screenshot of a computer program

Description automatically generated***

Im Backend Teil der OVWEB-Applikation sind ebenfalls alle Tests erfolgreich ausgeführt worden, wie im obigen Bild zu sehen ist.

## Auswerten

### Vergleich Ist/Soll

Der Vergleich zwischen dem Ist- und Soll-Zustand zeigt, dass die Einführung der Todesfallversicherungs-Prämienvergleiche erfolgreich war. Der Soll-Zustand forderte eine funktionelle und fehlerfreie Darstellung von bis zu drei Versicherungsangeboten auf der Detailseite, die gleichzeitig betrachtet und verglichen werden können. Die OVWEB-Anwendung wurde entsprechend angepasst und erfüllt die Anforderungen der IPA Aufgabe durch eine verbesserte Validierung und Fehleranzeige im Frontend.

### Verbesserungsvorschläge und Ideen

Im Verlauf der IPA kristallisierten sich diverse Ansätze heraus, die darauf abzielen, die Benutzerfreundlichkeit und Funktionalität der Anwendung signifikant zu steigern:

* **Markierung ausgewählter Offerten:** Nutzern sollte die Möglichkeit geboten werden, eine Offerte als „ausgewählt“ zu kennzeichnen, sobald eine Entscheidung für diese getroffen wurde. Diese Funktion würde die Übersichtlichkeit erhöhen und es den Nutzern erleichtern, ihre finale Auswahl zu verfolgen.
* **Versand von Offerten-Bundles per E-Mail:** Die Implementierung einer Option, mit der berechnete Offerten-Bundles direkt aus der Applikation heraus an Kunden per E-Mail versendet werden können, würde die Kommunikationseffizienz verbessern und den Nutzern ermöglichen, wichtige Informationen unverzüglich und sicher zu teilen.
* **Sortierung der Offerten nach Prämienhöhe:** Eine weitere sinnvolle Funktion wäre die Möglichkeit, Offerten nach der Höhe der Prämie zu ordnen. Dies würde den Vergleichsprozess für den Nutzer vereinfachen und eine gezielte Entscheidungsfindung unterstützen.
* **Ausblendung von Offerten im Bundle:** Nutzer sollten in der Lage sein, bestimmte Offerten in einem Bundle auszublenden. Diese Funktionalität würde es den Nutzern erlauben, sich auf die für sie relevanten Angebote zu konzentrieren und die Benutzeroberfläche nach individuellen Bedürfnissen anzupassen.
* **Überarbeitung erstellter Offerten:** Die Möglichkeit, einmal erstellte Offerten nachträglich bearbeiten zu können, stellt eine wesentliche Erweiterung dar. Dies würde es den Nutzern ermöglichen, auf Änderungen oder zusätzliche Anforderungen flexibel zu reagieren.

### Persönliches Fazit

In der Durchführung dieser IPA konnte ich meine fachspezifischen Kompetenzen in einem kontextbezogenen und anspruchsvollen Projektrahmen vertiefen und erweitern. Die Implementierung des Prämienvergleichs für Todesfallversicherungen integrierte meine tägliche berufliche Praxis mit zusätzlichen, komplexen Anforderungen. Besonders lag der Fokus auf der effizienten Umsetzung und Optimierung eines benutzerorientierten Vergleichstools innerhalb der OVWEB-Anwendung.

Der Entwicklungsprozess erforderte eine intensive Auseinandersetzung mit den funktionalen Anforderungen und eine agile Herangehensweise. Fortgeschrittene Validierungsmechanismen und die simultane Darstellung mehrerer Versicherungsangebote benötigten eine präzise technische Planung und Implementierung.

Im Zuge der Fehlerbehandlung und der Implementierung einer effektiven Benutzerführung wurde mein Verständnis für die technische Umsetzung von nutzerzentrierten Interfaces weiter geschärft. Nach Abschluss der Entwicklungsphasen wurden ausführliche Tests durchgeführt, die die Zuverlässigkeit und Stabilität der Software bestätigten. Diese Tests waren entscheidend dafür, dass jede Komponente gemäss den festgelegten Anforderungen funktionierte und die Software insgesamt robust gegenüber Fehlern war.

Ein weiterer kritischer Aspekt war die Dokumentation des Projekts. Die fortlaufende Anpassung der Software erforderte ständige Aktualisierungen der Dokumente, was den Prozess zeitintensiv und komplex machte. Diese Herausforderung unterstrich die Bedeutung einer sorgfältigen und dynamischen Projektdokumentation, um die Konsistenz und Nachvollziehbarkeit der Entwicklungsarbeit zu gewährleisten.

Insgesamt hat diese Projektarbeit meine Professionalität im Umgang mit komplexen Entwicklungsaufgaben bestätigt und meine Kompetenzen in der Anwendung moderner Softwareentwicklungstechniken verstärkt. Die erfolgreiche Umsetzung der geforderten Funktionalitäten innerhalb des eng gesteckten Zeitrahmens hat meine Fähigkeit unterstrichen, auch unter Druck präzise und effizient zu arbeiten. Die gewonnenen Erkenntnisse aus diesem Projekt werde ich in zukünftigen beruflichen Situationen nutzen, um die Qualität und Effizienz meiner Entwicklungsarbeit kontinuierlich zu steigern.

### Schlussreflexion

Diese IPA bot nicht nur die Möglichkeit, meine technischen Fähigkeiten zu erproben und zu erweitern, sondern diente auch als wertvolles Medium für persönliche Einsichten und professionelle Entwicklung. Die Reflexion über den gesamten Projektverlauf hat mir gezeigt, wie entscheidend Anpassungsfähigkeit und proaktives Problemmanagement in der dynamischen Welt der Softwareentwicklung sind.

Durch die Konfrontation mit unvorhergesehenen technischen Herausforderungen und die Notwendigkeit, ständig Lösungen zu adaptieren, habe ich ein tieferes Verständnis für die Komplexität von Softwareprojekten entwickelt. Diese Erfahrungen haben meine Fähigkeit gestärkt, flexibel auf Veränderungen zu reagieren und innovative Lösungswege zu erkunden. Ich habe gelernt, dass erfolgreiche Softwareentwicklung nicht nur von solidem technischen Know-how abhängt, sondern auch von der Fähigkeit, kreativ und resilient zu denken, auch wenn man unter Stress ist.

Die Dokumentation dieses Projekts spielte eine zentrale Rolle in der Bewertung meiner Arbeit und reflektierte den Fortschritt sowie die Herausforderungen, die während des Projektes auftraten. Diese umfasste nicht nur technische Details und Implementierungsschritte, sondern auch die Schwierigkeiten und Anpassungen, die notwendig waren, um die gesetzten Ziele zu erreichen. Die Notwendigkeit, meine Fortschritte und Überlegungen kontinuierlich zu dokumentieren, erwies sich als besonders herausfordernd, da die Dokumentation ständig aktualisiert werden musste, um die Änderungen im Projektverlauf genau festzuhalten.

Die gewonnenen Erkenntnisse aus dieser intensiven Arbeit haben mein professionelles Profil geschärft und werden mich als wertvolle Ressourcen in meiner weiteren Karriere begleiten. Diese Erfahrungen unterstreichen, wie wichtig es ist, stets lernbereit zu bleiben und sich den stetig ändernden Anforderungen der IT-Branche anzupassen.

## Glossar

|  |  |
| --- | --- |
| Begriff | Erklärung |
| ANGEBOTSoftware | Bisherige FAT-Client Software für die Verwaltung von Kunden und Angeboten der Generali durch Berater. |
| Business-Rules | Business Rules sind festgelegte Regeln oder Bedingungen, die bestimmen, welche Werte oder Aktionen in einem System zulässig sind. Sie sorgen dafür, dass nur logisch konsistente oder erlaubte Kombinationen von Daten oder Aktionen ausgeführt werden können, wie zum Beispiel die Festlegung, dass bestimmte Versicherungstypen Mindestbeträge oder Laufzeiten erfordern. |
| Detailseite | Die Seite im Frontend, auf der das berechnete Offerten-Bundle angezeigt wird. |
| Offerten-Bundle | Eine Todesfallversicherungs-Offerte mit mehr als eine Versicherungssumme. |
| OVWEB | Moderne cloudbasierte Webanwendung für die Verwaltung von Versicherungsangeboten der Generali. (**O**fferten**V**erwaltung **Web**) |
| Rechenkern der ANGEBOTSoftware | Der Teil der ANGEBOTSoftware, welches für die Berechnung und Validierung der Offerten zuständig ist. |
| shadcn-ui | Shadcn-UI ist eine UI-Komponentenbibliothek, die auf Tailwind CSS basiert und eine Reihe von vordefinierten, anpassbaren und responsiven Komponenten bietet, um die Entwicklung von  modernen Web-Oberflächen zu beschleunigen und zu  vereinfachen. |
| Zod | Zod ist eine TypeScript-Bibliothek zur Erstellung von Schemata  und Validierung von Daten, die es ermöglicht, Typsicherheit und  Datenvalidierung effizient zu kombinieren. |

## Quellenvezeichnis

|  |
| --- |
| **Name:** Flyway Database Migration Prefixes  <https://www.baeldung.com/database-migrations-with-flyway>  **Abrufdatum:** 08.04.2024  **Bemerkung:** Verwendet für Benennung von Flyway Migration Skripte |
| **Name:** ResponseStatusException Error Message  <https://stackoverflow.com/questions/62089792/how-to-display-error-message-into-a-json-using-responsestatusexception-on-spring>  **Abrufdatum:** 08.04.2024  **Bemerkung:** Verwendet für das Weiterleiten eines Fehlers durch ResponseStatusException. |
| **Name:** Set Error UseForm  <https://react-hook-form.com/docs/useform/seterror>  **Abrufdatum:** 08.04.2024  **Bemerkung:** Verwendet für das Setzen des ANGEBOTSoftware Fehlers auf der Create-Offer Seite. |
| **Name:** Tailwind CSS Style Documentation  <https://tailwindcss.com/docs/>  **Abrufdatum:** 09.04.2024, 10.04.2024  **Bemerkung:** Verwendet für Styling der neuen Komponenten für die Offerten-Detail-Seite. |
| **Name:** Grammatikkorrektur Software  <https://languagetool.org/de>  **Abrufdatum:** 17.04.2024  **Bemerkung:** Verwendet für Gramatik Korrektur der IPA. |
| **Name:** ChatGPT  <https://languagetool.org/de>  **Abrufdatum:** 17.04.2024  **Bemerkung:** Im Kapitel 15.1 finden sie eine genaue Beschreibung. |

### Einsatz von generativen Tools

Das generative Tool ChatGPT wurde in dieser Arbeit hauptsächlich zur Korrektur und zur Verfeinerung der Fachsprache eingesetzt. Es ist jedoch wichtig zu betonen, dass sämtliche Eingaben sorgfältig geprüft und die generierten Ausgaben überarbeitet und an den Projektbezug angepasst wurden. Es wurde darauf geachtet, dass keine Inhalte blind übernommen oder kopiert wurden.

**Verwendung:**

* Vorschläge für bessere Ausdrücke: ChatGPT wurde genutzt, um präzisere oder angemessenere Fachausdrücke zu finden, die die Klarheit und Professionalität des Textes verbessern.
* Grammatische Korrektur des Textes: Das Tool half dabei, Grammatikfehler zu identifizieren und zu korrigieren, um die sprachliche Qualität der Arbeit zu erhöhen.
* Formulierungshilfe für komplexe Sachverhalte: ChatGPT wurde eingesetzt, um komplexe technische Inhalte verständlich und korrekt zu formulieren.
* Optimierung der Textstruktur: Durch Vorschläge zur Textgliederung und zum logischen Aufbau konnte die Lesbarkeit und Nachvollziehbarkeit der Arbeit gesteigert werden.

Diese strategische Nutzung von ChatGPT hat dazu beigetragen, die Effizienz im Schreibprozess zu verbessern und gleichzeitig die inhaltliche und sprachliche Qualität der Arbeit sicherzustellen.

## Danksagung

Am Ende dieser Arbeit möchte ich allen danken, die zum Gelingen dieser IPA beigetragen haben. Ein besonderer Dank gilt Roman Babenko, meinem Fachexperten. Ohne ihn wäre der Abschluss der IT-Lehrlinge in der Generali unmöglich gewesen. Mein Dank gilt auch Daniel Sager und Roger Sorg für ihr Engagement zugunsten von uns Auszubildenden. Nicht zuletzt danke ich meinem Hauptexperten, Herrn Umachandran, für die investierte Zeit, um meine Arbeit zu prüfen, zu bewerten und mich während der Expertengespräche zu fördern.