**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

1. Projektaufbauorganisation 4

2. Organisation der Arbeitsergebnisse 5

2.1 Codesicherung 5

2.2 Dokumentationssicherung 5

3. Standards 6

4. Zeitplan 9

5. Arbeitsjournal 10

5.1 Tag 1: Mittwoch, 03.04.2024 10

5.2 Tag 2: Donnerstag, 04.04.2024 11

5.3 Tag 2: Freitag, 05.04.2024 11

6. Kurzfassung 13

6.1 Ausgangssituation 13

6.2 Projektvorgehen 13

6.3 Ergebnis 13

7. Projektvorgehen 14

Teil 2 – Individueller Teil Error! Bookmark not defined.

8. Informieren 15

8.1 Ziele der Aufgabenstellung 15

8.2 Vorgaben und Anforderungen 15

8.3 Ist-Zustand 15

8.3.1 Projekt-Überblick 16

8.3.2 Frontend 18

8.3.3 Backend 20

8.3.4 ERM-Datenbankdiagramm 26

8.3.5 ANGEBOTsoftware Rechenkern 27

8.3.6 Systemgrenzen 27

8.4 User Stories und Akzeptanzkriterien 27

8.4.1 Funktionale Anforderungen 27

8.4.2 Nichtfunktionale Anforderungen 28

9. Planen 30

9.1 Soll-Zustand 31

9.1.1 Frontendänderungen 32

9.1.2 Backendänderungen 33

9.2 Mockups 35

9.3 Testkonzept 37

9.3.1 Testziele und Strategie 37

9.3.2 Testumgebung 37

9.3.3 Testcases 38

10. Entscheiden 42

11. Realisieren 44

12. Kontrollieren 45

12.1 Testprotokoll 45

12.2 Testbericht 45

13. Auswerten 46

13.1 Vergleich Ist/Soll 46

13.2 Persönliches Fazit 46

13.2.1 Was lief gut? 46

13.2.2 Was lief schief? 46

13.2.3 Was habe ich aus der Erfahrung gelernt? 46

13.3 Schlussreflexion 46

14. Glossar 46

15. Quellenvezeichnis 46

15.1 Einsatz von generativen Tools 46

16. Danksagungen 46

1. 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

Um den Verlust der Dateien im Falle eines Ausfalles zu vermeiden, werden während der IPA verschiedene Versionierungs- und Speicherungssysteme genutzt.   
Mithilfe dieser Systeme kann ebenfalls auf ältere Versionen der Dokumente und Dateien zugegriffen werden.

### Codesicherung

Für den Code wird einen eigenständigen Branch in der Monorepo des Projekts erstellt. Änderungen am Code oder an der Projektdokumentation werden auf diesem Branch gepusht und somit gespeichert.

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

Folgend steht die Historie des Branches :

### Dokumentationssicherung

Für die Sicherung aller Dateien für die Dokumentation wird ein separates GitHub Repository erstellt. Auf diesem wird täglich ein Mal gepusht. Ebenfalls werden die Word-Dokumente automatisch auf OneDrive gespeichert.

**Link zum Repository:** https://github.com/Rad14nt/IPA\_2024\_Docs

Folgend steht die Historie des Repositorys :

## Standards

Im Ramen 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 |

|  |  |
| --- | --- |
| **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 und Informationsbeschaffung |
| Duden.de | Rechtschreibprüfung |

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

## Zeitplan

**A screenshot of a computer

Description automatically generated**

## Arbeitsjournal

### Tag 1: Mittwoch, 03.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * Aufgabenstellung und Anforderung studieren * Zeitplan erstellen * Ist-Zustand Analyse * User Stories und Akzeptanzkriterien definieren * Dokumentation – Teil 1, Administrativer Teil * Dokumentation – Informieren * - Dokumentation – Arbeitsjournal |
| **Probleme** | SendGrid stoppte das Backend vom starten, da mir die Authentifizierungsdaten im .env File fehlten. |
| **Hilfestellung** | Ich habe mein Arbeitskollege für die Authentifizierungsdaten für SendGrid gefragt, da er diese hatte und die Daten nicht online in der Dokumentation waren. |
| **Reflexion** | Ich konnte heute gut mit der IPA starten und kam auch sehr zügig voran, vor allem war die Vorlage besonders hilfreich. Den Zeitplan musste ich jedoch ganz neu gestalten, dies hat ein wenig mehr Zeit in Anspruch genommen als erwartet, aber ich konnte diese Zeit schnell mit dem Ausfüllen des administrativen Teils meines Dokumentes aufholen, bei der ich ein wenig schneller war als erwartet.  Weil ich mir für den Zeitplan jedoch Zeit genommen habe, erlaube dieser mir sehr strukturiert in meiner IPA einzusteigen. Etwas Stress gab es trotzdem, da ich noch am Anfang der IPA bin, ist alles noch ein wenig unsicher.  Ebenfalls hatte ich heute Probleme beim Aufsetzen meines Projektes, ich brauchte diesen aufzustarten, um Screenshots für den IST-Zustand zu machen, hatte aber Probleme mit einer neuen Implementation von SendGrid, ein E-Mail Service, welcher von jemand anders implementiert wurde. Es fehlten die Login Daten im Environment Variables File, und diese waren nicht in der Dokumentation zu finden, daher musste ich meinen Kollegen Samuel fragen, mir diese von seinem Projekt zu kopieren.  Allgemein bin ich aber mit dem heutigen Fortschritt zufrieden und freue mich, dass meine IPA so weit nach Plan läuft. |
| **Nächste Schritte** | * Informierphase abschliessen und fertig dokumentieren. * Planungsphase beginnen: * SOLL-Zustand definieren * Mockups erstellen * Planen dokumentieren |

### Tag 2: Donnerstag, 04.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * User Stories und Akzeptanzkriterien definieren * Dokumentation - Informieren * Soll-Zustand definieren inkl. Diagramme * Mockups erstellen (Figma) * Dokumentation - Planen * Dokumentation - Arbeitsjournal |
| **Reflexion** | Der heutige Tag lief genau nach Plan. Ich fing an mit das Definieren der User Stories und Akzeptanzkriterien. Dannach dauerte das Dokumentieren der Information Phase etwa eine halbe Stunde länger als erwartet. Weil ich mir aber schon Gedanken zur IPA gemacht hatte, konnte ich den Soll-Zustand schneller schreiben als gedacht.  Mittels Beispiele von andere Websiten konnte ich auch meine Mockups zeitgerecht fertig zeichnen und diese dannach auch fast komplett dokumentieren. Für die Dokumentation der Planung habe ich aber noch 2.5 Stunden, also sollte der Rest morgen auch gut machbar sein.  Hilfestellung brauchte ich heute keine und ich hatte auch keine Probleme.  Heute war ein erfolgreicher Tag an dem ich ohne viel Stress gut vorangekommen bin. |
| **Nächste Schritte** | * Model der Datenbank erstellen * Testkonzept erstellen * Planen dokumentieren * 1. Expertengespräch * Lösungsvariante festlegen * Entscheiden dokumentieren |

### Tag 3: Freitag, 05.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * Model der Datenbank erstellen * Testkonzept erstellen * Planen dokumentieren * 1. Expertengespräch * Lösungsvariante festlegen * Entscheiden dokumentieren * Dokumentation - Arbeitsjournal |
| **Ausserplanmässige**  **Arbeiten** | Ich hatte beim anschauen der Dokumentation gemerkt, dass das Response der ANGEBOTsoftware im Kapitel 8.3.3 nicht richtig war, ich musste dieses korrigieren. |
| **Reflexion** | Heute war ein entspannter Tag. Ich konnte schnell den Model der Datenbank erstellen und sprang dann direkt im Testkonzept, der dank meiner Akzeptanzkriterien schnell geschrieben wurde. Bei der Dokumentation, Teil Planen, brauchte ich ein wenig länger Zeit als erwartet, dies war weil ich gemerkt hatte, dass ich in der Dokumentation die falsche Schriftart für ein paar Kapiteln auswählte, ich musste dies korrigieren. Dies konnte ich mir aber leisten, da ich an der meine Lösungsvariante für die Detail Seite eine halbe Stunde schneller fertig hatte als gedacht. Heute war auch das erste Expertengespräch. Ich war recht nervös, aber weil meine Experten sehr freundlich waren konnte ich mich schnell beruhigen. Zuletzt musste ich dann noch die Lösungsvariante in meiner Dokumentation unter dem Kapitel Entscheiden dokumentieren, und schrieb dann zuletzt noch diese Reflexion. |
| **Nächste Schritte** | Realisieren - Backend:   * Datenbank Anpassungen * Anpassungen Model und DTOs * Anpassungen an Controller und Services * Realisieren dokumentieren |

### Tag 4: Montag, 08.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * Datenbank Anpassungen * Anpassungen Model und DTOs * Anpassungen an Controller und Services * Realisieren dokumentieren * Frontend : Anpassungen an der Create-Offer Seite * Dokumentation - Arbeitsjournal |
| **Ausserplanmässige**  **Arbeiten** | * Frontend : Anpassungen an der Create-Offer Seite |
| **Reflexion** | Ich startete heute mit der Realisierugnsphase meiner IPA. Am morgen ging es direkt los mit den Anpassungen an der Datenbank. Damit meine Datenbank mit meine Applikation harmonisiert musste ich dannach Models, DTOs, Mappers und Tests an den neuen Änderungen anpassen. Somit mussten dann auch die Controllers und Services leicht angepasst werden.  In meiner IPA stehen zwei User Aktivitäten im Vordergrund. Die Validierung einer Offerte auf der Erstellungsseite und das Offertenvergleich auf der Detail Seite. Ich implementierte heute die Logik für beide in meinen Backend. Das Error Handling funktioniert jedoch nicht perfekt.  Grund dafür ist eine kleine Abweichung vom Zeitplan.  Nach dem ich die Möglichkeit hinzufügte, eine Offerte mit mehreren Vericherungssummen zu erstellen und diese zu validieren im Backend implementierte, habe ich entschieden, dass es Sinn machen würde, wenn ich die Create-Offer Seite auch schon anpasse. Dies damit ich meine Applikation auch gerade testen konnte, ohne Postman zu benutzen.  Eine kurze Implementation war auch schnell erledigt. Ich fand dann heraus, dass die Implementation tatsächlich auch funktionierte. Somit muss ich das Error Handling im Backend Teil des Projektes aber am Dienstag oder am Mittwoch realisieren.  Zum Schluss implementierte ich auch noch, dass das Backend gleich drei Offerten berechnen kann, mittels Network Tool auf dem Browser konnte ich bestätigen, dass dies auch funtkionierte. Dies war schnell erledigt. Am Schluss vom Tag dokumentierte ich das Erreichte. |
| **Nächste Schritte** | * Anpassungen an Controller und Services * Anpassungen an der Create-Offer Seite * Realisieren dokumentieren |

### Tag 5: Dienstag, 09.04.2024 - Halbtag

|  |  |
| --- | --- |
| **Tätigkeiten** | * Frontend : Anpassungen an Controller und Services * Frontend : Anpassungen an der Create-Offer Seite * Frontend : Anpassungen an der Offer-Detail Seite   - Dokumentation - Arbeitsjournal |
| **Ausserplanmässige**  **Arbeiten** | * Frontend : Anpassungen an der Offer-Detail Seite * Anpassungen an Controller und Services |
| **Hilfestellung** | * Tailwind CSS Dokumentation |
| **Reflexion** | Heute war mein erster Halbtag. Nach einem etwas stressigen Mittagessen kam ich in die Arbeit und legte los. Ich entschied heute bewusst, nicht zu dokumentieren. Dies war aus dem Grunde, dass ich entschieden habe die Anpassungen an Controller und Services zu verschieben, da es immer wieder Anpassungen brauchen würde, die ich erst merkte, als ich das Frontend implementierte. Ich legte heute eigentlich direkt mit den Controller und Services los, ich merkte am Tag zuvor, dass das Fehlerhandling bei der Validierung noch ungenügend war. Deshalb implementierte ich neues Fehlerhandling. Nach dem programmierte ich die noch kleinen notwendigen Änderungen im Frontend auf der Erstellungsseite, die Erstellungsseite war somit schneller fertig als erwartet, um etwa eine Stunde. Ich machte dann weiter mit der Offer-Detail Seite. Dort kam ich auch schnell vorwärts. Ich implementierte zuerst die Möglichkeit, eine Liste von Offerten zu empfangen. Nachher machte ich mich ans Darstellen der Offerten. Ich schrieb ein neues Component, welches ich auf der Detail Seite einsetzte. Um diesen Component zu schreiben musste ich die Dokumentation von TailwindCSS zur Hilfe nehmen, damit ich die Offerten auch nach meinem Figma Design stylen konnte. Ich schaute ebenfalls darauf, dass der neue Komponente reusable ist. Nachdem ich den Komponenten fertig hatte, merkte ich, dass obwohl die Applikation funktionierte, noch viele Fehler auf der Detail Seite jetzt erschienen. Diese muss ich dann am folgenden Tag beheben. Auch muss das Fehlerhandling im Controller und Services für die Berechnungen geschrieben werden, so wie im Frontend. |
| **Nächste Schritte** | * Anpassungen an der Offer-Detail Seite * Anpassungen an Controller und Services * Implementierung Logging * Realisieren dokumentieren |

### Tag 6: Mittwoch, 10.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * Anpassungen an der Offer-Detail Seite * Anpassungen an der Create- Offer Seite * Anpassungen an Controller und Services * Implementierung Logging * Realisieren dokumentieren * Arbeitsjournal schreiben |
| **Ausserplanmässige**  **Arbeiten** | * Unit Tests schreiben |
| **Hilfestellung** | * Tailwind CSS Dokumentation * React-Hook-Form Dokumentation |
| **Reflexion** | Den heutigen Tag find ich heute mit der Fertigstellung der Erstellungsseite an. Es mussten noch ein paar Änderungen an dem Fehlerhandling vorgenommen werden, nämlich eine leichte Korrektur an der Art der Fehlerhandling. Weil ich nicht genau wusst wie die Implementation aussah, nahm ich die Reach-Hook-Form Dokumentation zur Hilfe. Ich machte dann mit der Umsetzung der Detail Seite weiter. Heute war das Ziel das Design so nah wie möglich an dem Figma Mockup zu bringen. Dafür benutzte ich wieder die Dokumentation von Tailwind CSS. Als ich das Testen wollte merkte ich aber, dass mein Backend nicht mehr richtig lief. Dies war aus dem Grund, dass die schon bestehende Tests fehlgeschlagten, weil diese nicht mit den neuen Versicherungssummen versehen wurde. Dies musste repariert werden, somit musste ich schon jetzt eine halbe Stunde in den Tests einsetzen. Nachdem dies repariert war entwickelte ich im Backend weiter. Ich musste ein Model anpassen, damit meine Validierung im Frontend richtig funktionierte. Nach all diesen Änderungen wollte ich dokumentieren, es machte Sinn dies zu tun, bevor ich mit dem Logging und Tracing anfangen müsste. Ich merkte aber, dass es bei der Dokumentation mehr Aufwand braucht, Sie sinnvoll zu schreiben, damit eine Person die nichts mit dem Projekt zu tun hat diese auch verstehen kann, als gedacht.  Zuletzt implementierte ich ein wenig am Logging in den neuen Funktionen.  Ich merkte dabei, dass viel Logging und Tracing schon implementiert ist. Im Rahmen dieser IPA werden viele Methoden umgeschrieben, und nicht neugeschrieben, aus dem Grund scheint viel im Tracing und Logging schon gemacht zu sein und wird wahrscheinlich weniger dauern als gemeint. Dies ist aber gut, weil ich denke, dass ich mehr Zeit an der Dokumentation brauchen muss. |
| **Nächste Schritte** | * Realisieren Dokumentieren * Implementierung Tracing * Implementierung Logging * Unit Tests schreiben * 2. Experteninterview |

Teil 2 – Indivudueller 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, zu welcher Sie mehr Informationen in Kapitel 7 finden können. Nach einer eingehenden Informations- und Planungsphase, in der der aktuelle Zustand erfasst und der gewünschte Zustand definiert wird, ermöglicht dies den Start der Umsetzungsphase mit einem klaren Verständnis der Ziele. 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 werden die Detailansicht 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

## 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. Ebenfall werden gesammelten Erfahrungen analyisiert, um sowohl die Stärken aus auch die Verbesserungspotenzialle des Projektes 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, egal ob sie die Geschäftsregeln verletzt oder nicht.   
Nach der Erstellung erfolgt eine Weiterleitung im Frontend zur Detailseite mit der Offerten-ID. Dort wird erneut eine Abfrage an die Datenbank 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.A diagram of a software

Description automatically generated

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

Das Diagramm dient der Übersicht und ist korrekt, beschreibt aber nicht jedes Detail das passiert.

A diagram of a company

Description automatically generated with medium confidence

#### Frontend

Folgend befinden sich die IPA-relevante Seiten.

**A white background with red and blue text

Description automatically generatedOffer Dashboard**

**A white paper with red and blue lines

Description automatically generatedOffer Creation**

**A screenshot of a computer

Description automatically generatedOffer Details – Erfolgreiche Berechnung der Offerte**

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

Description automatically generated**

#### 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 Offerten gibt es Controller und Services für die Verwaltung der Offerten, sowie auch einen Controller und einen Service für die Berechnung der Offerten.*

**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.   
Auch kann sich die Berechnung der Offerte kann sich jedoch ändern, beispielsweise bei einer Prämienerhöhung im Jahr 2025. Aus diesem Grund wird die Offerte bei jedem Aufruf der Detailseite neu kalkuliert, anstatt sie dauerhaft kalkuliert zu speichern.

**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 customerId Variable gesetzt und dann dem Service mit dem obigen Offertendaten aus dem DTO weitergeleitet, dort werden die Kundendaten mittels obig genannten ID aus der Datenbank herausgesucht. Ebenfalls wird die ID des Beraters aus der Sponsor Tabelle mittels Keycloak ID herausgesucht. Das Keycloak Id des Beraters wird mit dem JWT auch dem Controller übergeben.

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.

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

Zusammen mit den Offertendaten aus dem DTO und der Sponsor-ID werden die Kundendaten in diesem Modell zugeordnet. Dafür nutzen wir die Methode mapToLifeInsuranceRequest, 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 mit Hilfe der externen ANGEBOTSoftware berechnet. Diese Berechnung kann, wie in Kapitel 8.3.2 dargestellt, bei einer Verletzung der Geschäftsregeln 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 im Repository 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:**

A screenshot of a computer code

Description automatically generated

A white background with black text

Description automatically generated**Response:**

Diese Response könnte jedoch auch ein Fehler zurückweisen, dies würde wie folgend aussehen:

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

#### ANGEBOTsoftware Rechenkern

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.

#### Systemgrenzen

Diese Arbeit zielt auf die Erstellung, Berechnung, Validierung und Anzeige mehrer Todesfallversicherungsofferten. Daher gehört das ANGEBOTSoftware, spezifisch das Rechenkern davon, als Abhängigkeit zu dieser Aufgabe.

Es ist wichtig zu erwähnen, dass SendGrid, welches für E-Mails implementiert ist, und Keycloak, nicht Teil der Systemgrenzen dieser Aufgabe sind.

### 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. |

|  |
| --- |
| **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 nachvollziehen 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 Komponente sind aus kleinere, wiederverwendbare Komponente zusammengebaut. |
| **AC 8.2:** Diese Elemente müssen in der Storybook dokumentiert werden. |

## Planen

### Soll-Zustand

Der Soll-Zustand sollte es dem Berater erlauben, maximal drei Versicherungssummen einzugeben, um einen Vergleich der Prämien dem Kunden zu präsentieren.

Ebenfalls sollte eine Validierung bei der Erstellung dieser neue Offerte passieren.

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

Damit das neue Prozess einfacher zu verstehen ist, wurde ein das im Kapitel 8.3.1 gezeichnete Diagramm mit den neuen geplannten Änderungen angepasst.

A close-up of a document

Description automatically generated

#### 

#### Frontendänderungen

Um die Änderungen visuell zu sehen, finden Sie in Kapitel 9.2 die dazugehörige Mockups. Die neuen Komponenten werden der Mockups entsprechen.

**Storybook Anpassungen**

Alle neue Komponenten müssen im Storybook erfasst werden.

**Tracing**

Für jedes Request wird eine Tracing ID genertiert.

**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 einem 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, so dass diese die Offerte zuerst validieren, bevor diese abgespeichert wird, die Methode dafür wird im LifeInsuranceCalculationService implementiert, da sie in direkten 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 können wir in der neue Validierungsmethode die schon bestehende Methode calculateOffer aus dem LifeInsuranceCalculationController benutzen.

**Model und DTO Änderungen**

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

**Unittests**

Wo möglich müssen automatisierte Unit Tests eingesetzt werden.

### Mockups

A screenshot of a computer

Description automatically generated**Offer Creation**

A screenshot of a computer

Description automatically generated**Offer Creation – Fehlermeldung**

A screenshot of a computer

Description automatically generated**Offer Details – Variante 1**

A screenshot of a computer

Description automatically generated**Offer Details – Variante 2**

### 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 funktionale 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 Unit-Tests.   
Fallls dies nicht möglich ist, werden basierend auf die User Stories und Akzeptanzkriterien manuelle Tests eingesetzt.

#### 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** | Berater kann eine Offerte mit bis zu 3 Versicherungssummen erstellen. |
| **Voraussetzung** | Der Berater ist eingeloggt und hat auf der Offerten Erstellungsseite navigiert. |
| **Schritte** | Berater füllt das Formular aus, und gibt eine bis drei Versicherungssummen an. |
| **Erwartetes Resultat** | Die Applikation validiert die Eingaben (Frontend-Seitig) |

|  |  |
| --- | --- |
| **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 Detail Seite weitergeleitet. Ebenfalls kann von der Offerten-Dashboardseite aus auf eine Offerte klicken, um auf die Detail Seite zu kommen. |
| **Erwartetes Resultat** | Die Offerte wird geholt. 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 Detail Seite weitergeleitet wie zuvor * Dem Berater wird das von der ANGEBOTSoftware angegebene Fehler angezeigt |

|  |  |
| --- | --- |
| **Testcase 4** | |
| **Anforderung:** | Story 4, AC 4.1 – 4.2 |
| **Beschreibung** | Benutzerfreundlichkeit der Detail Seite |
| **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 Dateien. |
| **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 ID’s 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 Testabdeckung und der Effektivität der Tests für die neu implementierte Funtkionen, sowohl im Frontend als auch im Backend. |
| **Voraussetzung** | Neue Funktionen wurden hinzugefügt und Tests wurden geschrieben. |
| **Schritte** | Durchführung der neu erstellten Tests. |
| **Erwartetes Resultat** | Die Tesfälle decken alle neue Funktionen ab. UnitTests 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. |

### Kriterien für die Entscheidungsphase

In diesem Kapitel werden die Kriterien und deren Gewichtung für die Entscheidungsphase definiert. Es wird entschieden, welches der zwei Varianten der Offer Details Seite im Kapitel 9.2 implementiert werden sollte. Jede Variante sollte eine Note zwischen 1 und 5 für jedes Kriterium bekommen.

|  |  |  |
| --- | --- | --- |
| **Kriterium** | **Gewichtung** | **Begründung** |
| **Benutzerfreundlichkeit** | 0.4 | Die Benutzbarkeit steht im Vordergrund, da dies einer Kriterium der IPA ist. Es ist wichtig, dass die Prämien zum Vergleich im Vordergrund stehen. |
| **Komplexität** | 0.3 | Wenn die Implementation einfacher ist, bleibt mehr Zeit für andere Aufgaben. |
| **Ästhetik** | 0.3 | Die Ästhetik spielt nicht die grösste Role, daher wird sie auch weniger gewichtet. Sie ist aber trotzdem wichtig für eine gute Benutzererfahrung und Klarheit. |

## Entscheiden

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Variante 1** | **Variante 2** |
| **Kriterium** | **Gewichtung** | **Note** | **Note** |
| Benutzerfreundlichkeit | 0.4 | 5.5 | 4.5 |
| Komplexität | 0.3 | 4 | 4 |
| Ästhetik | 0.3 | 5.5 | 4.5 |
| **Note:** | | **5.1** | **4.4** |

**Begründung der Noten**

Benutzerfreundlichkeit:

In der Variante 1 steht die Prämie mehr im Vordergrund, da alle andere Daten viel kleiner sind. In der zweiten Variante steht sie zwar auch im Vordergrund, da die andere 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.

Weil wir Tabellen vom Componenten Library Shadcn UI benutzen, kann es aus Erfahrung recht schwierig sein Variante 2 umzusetzen.

Ästhetik:

Die Ästhetik der ersten Variante ist viel schöner und moderner. Somit wird sie besser bewertet als Variante 2.

## Realisieren

### Datenbank

Es wurden wenige Änderungen an der Datenbank vorgenommen.   
Wie im vorherigen Kapitel geplannt, 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.

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

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

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

Durch folgendes 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

Wegen den Änderungen an der Datenbank und um die neue Funktionalitäten umzusetzen, mussten existente Models und DTOs angepasst werden. Die bestehende „vs“ Variable musste mit eine „vs1“ Variable getauscht werden, und es mussten zwei neue Variabeln, vs1 und vs2 hinzugefügt werden. Ebenfalls mussten die Models und DTOs umgeschrieben werden, um die Validierung im Frontend zu ermöglichen.

**LifeInsuranceRequestWithCustomerInfo Model**

Dieser Dto enthaltet alle Information die für die Berechnung einer Offerte erforderlich sind. Er enspricht der Tabelle in der Datenbank.

Hier mussten die Variabeln angepasst werden, um die neue Versicherungssummen zu erlauben.

Weil es sein könnte, dass ein Offerten-Bundle aber nur eine Offerte enthaltet und somit nur eine Versicherungssumme, werden die neue Variabeln vs1 und vs2 als **nullable = true** markiert. Die Annotation @Column verbindet die Variable 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 ausser die @Column Annotationen die gleichen Änderungen wie im obigen Model vorgenommen werden, um die vs1, vs2 und vs3 Variabeln zu implementieren.

Die Annotation @NotNull beschreibt, dass die Variable nicht leer sein darf.

@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 Detail Seite.

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 Teilen.

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

Der Status, die Offertendetails und ein Error Code.   
Der Status und der Error Code werden mit jede 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 OffertenDetails beinhaltet alle Daten zur erfolgreich berechneten Offerte, dieser wurde im Rahmen dieser IPA neu erstellt. Zuvor befangen sich alle Daten der OfferDetails direkt im LifeInsuranceOfferDto.

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.  
Alle Variabeln, die im Frontend auf der Detail Seite angezeigt werden und sich jetzt innerhalb der OfferDetail befinden, wurden mit @NotNull markiert, um sicherzustellen, dass auch alles richtig übergeben wird.

Die error beinhaltet interne Fehler in der OVWEB Applikation.

**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);

Die offerDetail Variable wird dann direkt dem LifeInsuranceOfferDtos übergeben, wo die jetzt im offerDetail erhaltene Daten jetzt nicht mehr direkt übergeben werden, sondern über die neue offerDetail Variable.

return new LifeInsuranceOfferDto(  
 LifeInsuranceOfferDto.Status.*fromValue*(calculation.getStatus().getValue()),  
 offerDetail,  
 calculation.getError().stream().map(com.generali.fosoft.model.Error::getValue).toList());  
}

#### Controller und Services

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

Wie im Kapitel 8.3.3 beschrieben, sind die Controllers und Services wie folgend strukturiert:

LifeInsuranceRequest –> Verwaltung der Offerten

LifeInsuranceCalculation -> Berechnung der Offerten

**LifeInsuranceRequestController**

In der LifeInsruanceRequestController 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));

Wir geben der checkOfferValidity Methode den request ab, zusammen mit der customerId. Der Request wird dann an der notwendigen Struktur gemappt. In diesem Fall zu einem LifeInsuranceRequestWithCustomerInfo Model.

Die Antwort ist eine Liste von ProductCalculationResponses.

Nachdem wir die Antwort erhalten, iterieren wir über die Liste und verifizieren die Variable Status.

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 das beschriebene Fehler zurückgewiesen.

**LifeInsuranceCalculationService**

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

*checkOfferValidity*

Weil wir aus dem Controller die Customer ID mitgeben und die Kundendaten für die Validierung notwendig sind, holen wir diese zuerst aus der Datenbank. 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 uns die Kundendaten zurück.

Weil wir jetzt womöglich ein Offerten-Bundle validieren müssen, das heisst mehrere Berechnungen weil es mehr als eine Versicherungssumme gibt, holen wir die Versicherungssummen aus dem Request heraus und speichern sie in einer Liste.

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

Mithilfe dieser Liste kann jetzt iteriert werden und somit für jede Versicherungssumme eine Validierung gemacht werden, je nachdem wie viel Versicherungssummen vorhanden sind. Die Validierung findet mittels der calculateLifeInsurance Methode stadt. Diese antwortet entweder mit eine berechnete Offerte, oder mit einem Fehler. Wir speichern jede dieser Responses in einer Liste von Typ ProductCalculationResponse.

#### Observability Logging

### Frontend

### Tracing

## Kontrollieren

### Testprotokoll

### Testbericht

Beschreiben was bewusst nicht getestet wurde.

## Auswerten

### Vergleich Ist/Soll

### Persönliches Fazit

#### Was lief gut?

#### Was lief schief?

#### Was habe ich aus der Erfahrung gelernt?

### Schlussreflexion

## Glossar

## Quellenvezeichnis

Flyway db migration naming convention

[Database Migrations with Flyway | Baeldung](https://www.baeldung.com/database-migrations-with-flyway)

Rsc show message always

<https://stackoverflow.com/questions/62089792/how-to-display-error-message-into-a-json-using-responsestatusexception-on-spring>

styling

<https://tailwindcss.com/docs/>

Error handling on creation site

https://react-hook-form.com/docs/useform/seterror

### Einsatz von generativen Tools

## Danksagungen