**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 9

3. Standards 10

4. Zeitplan 13

5. Arbeitsjournal 14

5.1 Tag 1: Mittwoch, 03.04.2024 14

5.2 Tag 2: Donnerstag, 04.04.2024 14

5.3 Tag 3: Freitag, 05.04.2024 15

5.4 Tag 4: Montag, 08.04.2024 16

5.5 Tag 5: Dienstag, 09.04.2024 - Halbtag 17

5.6 Tag 6: Mittwoch, 10.04.2024 18

5.7 Tag 7: Donnerstag, 11.04.2024 19

6. Kurzfassung 21

6.1 Ausgangssituation 21

6.2 Projektvorgehen 21

6.3 Ergebnis 21

7. Projektvorgehen 22

8. Informieren 23

8.1 Ziele der Aufgabenstellung 23

8.2 Vorgaben und Anforderungen 23

8.3 Ist-Zustand 23

8.3.1 Projekt-Überblick 24

8.3.2 Frontend 26

8.3.3 Backend 28

8.3.4 ERM-Datenbankdiagramm 35

8.3.5 ANGEBOTsoftware Rechenkern 36

8.3.6 Systemgrenzen 36

8.4 User Stories und Akzeptanzkriterien 36

8.4.1 Funktionale Anforderungen 36

8.4.2 Nichtfunktionale Anforderungen 37

9. Planen 40

9.1 Soll-Zustand 40

9.1.1 Frontendänderungen 42

9.1.2 Backendänderungen 43

9.2 Mockups 45

9.3 Testkonzept 47

9.3.1 Testziele und Strategie 47

9.3.2 Testumgebung 47

9.3.3 Testcases 48

9.4 Kriterien für die Entscheidungsphase 51

10. Entscheiden 52

11. Realisieren 53

11.1 Datenbank 53

11.2 Backend 54

11.2.1 Änderungen an Models, DTOs und Mappers 54

11.2.2 Controller und Services 57

11.3 Frontend 65

11.3.1 Actions 65

11.3.2 ProductCalculationDisplay Component 68

11.3.3 Seiten 69

11.4 Automatisiertes Testing 72

11.4.1 Backend Unit Tests 72

11.4.2 Frontend Jest Tests 72

12. Kontrollieren 73

12.1 Testprotokoll 73

12.2 Testbericht 73

13. Auswerten 74

13.1 Vergleich Ist/Soll 74

13.2 Persönliches Fazit 74

13.2.1 Was lief gut? 74

13.2.2 Was lief schief? 74

13.2.3 Was habe ich aus der Erfahrung gelernt? 74

13.3 Schlussreflexion 74

14. Glossar 74

15. Quellenvezeichnis 74

15.1 Einsatz von generativen Tools 74

16. Danksagungen 75

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 :

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 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 verhinderte den Start des Backends, da die Authentifizierungsdaten im .env-File fehlten. |
| **Hilfestellung** | Ich habe meinen Arbeitskollegen nach den Authentifizierungsdaten für SendGrid gefragt, da er diese besass und die Daten nicht online dokumentiert waren. |
| **Reflexion** | Der Start in die IPA verlief heute recht zügig und war besonders durch die vorhandene Vorlage erleichtert. Allerdings musste ich den Zeitplan komplett neu gestalten, was mehr Zeit in Anspruch nahm als erwartet. Diese konnte ich jedoch durch schnelleres Arbeiten am administrativen Teil meines Dokumentes ausgleichen. Die sorgfältige Erstellung des Zeitplans ermöglichte mir einen strukturierten Einstieg in die IPA, auch wenn zu Beginn etwas Stress aufkam, da noch Unsicherheiten bestanden.  Zudem gab es heute Schwierigkeiten beim Aufsetzen meines Projektes. Ich musste es starten, um Screenshots für den IST-Zustand zu machen, stiess jedoch auf Probleme mit einer neuen Implementierung von SendGrid, einem E-Mail-Service, der von jemand anderem implementiert worden war. Es fehlten die Login-Daten in der Umgebungsvariablen-Datei, die nicht in der Dokumentation vermerkt waren, weshalb ich meinen Kollegen Samuel bitten musste, mir diese aus seinem Projekt zu kopieren.  Insgesamt bin ich mit dem Fortschritt des heutigen Tages zufrieden und freue mich, dass meine IPA weitgehend nach Plan verläuft. |
| **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 verlief genau nach Plan. Ich begann damit, die User Stories und Akzeptanzkriterien zu definieren. Das Dokumentieren der Informationsphase dauerte etwa eine halbe Stunde länger als erwartet. Da ich mir jedoch bereits im Vorfeld Gedanken zur IPA gemacht hatte, konnte ich den Soll-Zustand schneller als geplant fertigstellen. Mit Hilfe von Beispielen anderer Websites gelang es mir auch, meine Mockups rechtzeitig zu zeichnen und fast vollständig zu dokumentieren. Für die Dokumentation der Planung habe ich noch 2,5 Stunden eingeplant, sodass der Rest morgen ebenfalls gut zu bewältigen sein sollte.  Heute benötigte ich keine Unterstützung und hatte auch keine Schwierigkeiten. Es war ein erfolgreicher Tag, an dem ich ohne viel Stress gute Fortschritte machen konnte. |
| **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 dokumentiert war, ich musste dieses korrigieren. |
| **Reflexion** | Heute war ein recht entspannter Tag. Zunächst stellte ich beim Durchsehen der Dokumentation fest, dass die Antwort der ANGEBOTsoftware in Kapitel 8.3.3 fehlerhaft war, was ich korrigieren musste. Anschliessend konnte ich zügig das Datenbankmodell erstellen und direkt zum Testkonzept übergehen, das sich dank meiner Akzeptanzkriterien schnell formulieren liess.  Bei der Dokumentation im Planungsteil benötigte ich etwas mehr Zeit als erwartet. Der Grund dafür war, dass ich feststellte, dass ich in einigen Kapiteln die falsche Schriftart verwendet hatte, was eine Korrektur erforderlich machte. Diese zusätzliche Zeit konnte ich mir jedoch leisten, da ich die Lösungsvariante für die Detailseite eine halbe Stunde schneller fertigstellen konnte als geplant.  Heute fand auch mein erstes Expertengespräch statt. Obwohl ich anfangs recht nervös war, half mir die freundliche Art meiner Experten schnell, mich zu beruhigen. Zum Abschluss des Tages musste ich noch die Lösungsvariante im Kapitel "Entscheiden" meiner Dokumentation festhalten und verfasste 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** | Heute begann ich mit der Realisierungsphase meiner IPA. Der Tag startete direkt mit Anpassungen an der Datenbank. Anschliessend musste ich Modelle, DTOs, Mapper und Tests an die neuen Änderungen anpassen, was auch leichte Modifikationen an den Controllern und Services erforderlich machte.  In meiner IPA stehen zwei Nutzeraktivitäten im Mittelpunkt: die Validierung einer Offerte auf der Erstellungsseite und der Offertenvergleich auf der Detailseite. Ich implementierte heute die Logik für beides in meinem Backend, allerdings funktioniert das Fehlermanagement noch nicht einwandfrei. Dies liegt an einer kleinen Verzögerung im Zeitplan.  Nachdem ich die Funktion hinzugefügt hatte, eine Offerte mit mehreren Versicherungssummen zu erstellen und im Backend zu validieren, entschied ich mich, auch die Seite zur Angebotserstellung anzupassen. Dies ermöglichte es mir, meine Applikation direkt zu testen, ohne Postman verwenden zu müssen. Die Implementierung war kurz und erfolgreich. Ich stellte fest, dass die Implementierung funktionierte, weshalb das Fehlermanagement im Backend-Teil des Projekts jedoch erst am Dienstag oder Mittwoch realisiert werden muss.  Zum Abschluss implementierte ich auch noch die Fähigkeit des Backends, gleich drei Offerten zu berechnen. Mittels des Netzwerk-Tools im Browser konnte ich bestätigen, dass dies auch funktionierte. Am Ende des Tages dokumentierte ich die erzielten Fortschritte. |
| **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 startete ich meinen Arbeitstag nach einem etwas stressigen Mittagessen und entschied mich bewusst gegen die Dokumentation. Der Grund dafür war, dass ich geplante Anpassungen an Controllern und Services aufgrund kontinuierlicher Änderungsbedarfe, die mir erst während der Implementierung des Frontends auffielen, verschob. So begann ich direkt mit den Controllern und Services, insbesondere da ich am Vortag bemerkt hatte, dass das Fehlerhandling bei der Validierung noch unzureichend war. Daher implementierte ich ein verbessertes Fehlerhandling.  Anschliessend nahm ich noch notwendige Änderungen auf der Erstellungsseite vor, die dadurch schneller als erwartet fertiggestellt war – etwa eine Stunde früher, es würde jedoch noch später kleine Anpassungen brauchen. Danach widmete ich mich der Offer-Detailseite, auf der ich ebenfalls rasche Fortschritte machte. Zuerst implementierte ich die Möglichkeit, eine Liste von Offerten zu empfangen, und begann dann, die Offerten darzustellen. Ich entwickelte ein neues, wiederverwendbares Component, das ich auf der Detailseite einsetzte. Für die Gestaltung dieses Components nutzte ich die Dokumentation von TailwindCSS, um sicherzustellen, dass die Offerten entsprechend meinem Figma-Design gestylt werden konnten.  Nachdem ich den Komponenten fertiggestellt hatte, stellte ich fest, dass das Design noch nicht pixel-genau mit dem Figma-Design übereinstimmte. Diese Probleme muss ich am folgenden Tag angehen. Zusätzlich steht noch das Schreiben des Fehlerhandlings im Controller und in den Services für die Berechnungen aus. |
| **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** | Heute begann ich mit der Fertigstellung der Erstellungsseite und musste einige Anpassungen am Fehlerhandling vornehmen, insbesondere eine leichte Korrektur an der Art des Fehlermanagements. Da ich mir über die genaue Implementierung unsicher war, griff ich auf die React-Hook-Form Dokumentation zurück. Anschliessend setzte ich die Arbeit an der Detailseite fort, wobei mein Ziel war, das Design möglichst nah an das Figma Mockup anzupassen, wofür ich erneut die Tailwind CSS Dokumentation verwendete.  Beim Testen stellte ich fest, dass mein Backend nicht korrekt funktionierte, weil die bestehenden Tests aufgrund der neuen Versicherungssummen fehlschlugen. Dies erforderte sofortige Reparaturen, weshalb ich eine halbe Stunde in die Korrektur der Tests investieren musste. Nachdem dies behoben war, entwickelte ich weiter am Backend und musste ein Modell anpassen, um die Validierung im Frontend korrekt durchzuführen.  Nach all diesen Änderungen plante ich zu dokumentieren. Es erschien sinnvoll, dies zu tun, bevor ich mit dem Logging und Tracing beginnen würde. Jedoch stellte ich fest, dass das Verfassen einer verständlichen mehr Aufwand erforderte als angenommen.  Zum Schluss arbeitete ich an der Implementierung des Loggings in den neuen Funktionen. Dabei bemerkte ich, dass bereits viel Logging und Tracing implementiert war. Da im Rahmen dieser IPA viele Methoden umgeschrieben und nicht neu geschrieben werden, scheint vieles im Bereich Tracing und Logging bereits erledigt zu sein, was wahrscheinlich weniger Zeit in Anspruch nehmen wird als zunächst gedacht. Dies ist vorteilhaft, da ich vermute, dass ich mehr Zeit für die Dokumentation benötigen werde. |
| **Nächste Schritte** | * Implementierung Tracing * Implementierung Logging * Unit Tests schreiben * 2. Experteninterview |

### Tag 7: Donnerstag, 11.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * Implementierung Tracing * Implementierung Logging * Tests schreiben * 2. Experteninterview * Arbeitsjournal schreiben. |
| **Ausserplanmässige**  **Arbeiten** | * Realisieren Dokumentieren |
| **Reflexion** | Heute war wirklich ein erfolgreicher Tag. Ich habe die Implementierungsphase erfolgreich abgeschlossen, indem ich die letzten Tests schrieb und meine finalen Commits pushte. Das Tracing und Logging nahmen weniger Zeit in Anspruch als erwartet, jeweils eine Stunde weniger, wodurch ich auch mit den Tests bereits heute fertig werden konnte. Das bedeutet, dass ich morgen mehr Zeit für die Dokumentation zur Verfügung haben werde, was ich auch dringend benötige.  Zusätzlich fand heute das zweite Expertengespräch statt, welches schneller abgeschlossen war als erwartet. Nach einer Besprechung mit meinem Hauptexperten änderte ich im Zeitplan die Beschreibung der Tests von „Unit Tests“ zu „Tests“, um klarzustellen, dass damit sowohl Frontend- als auch Backend-Tests gemeint sind, dies war nur ein kleines Defintionsfehler, jedoch wollte ich sicherstellen, dass klar verstanden ist, was gemeint ist.  Ich hatte heute keinerlei Probleme und der Tag verlief weitgehend nach Plan. Für morgen steht die Dokumentation auf dem Programm, einschliesslich der Durchführung und Dokumentation der automatisierten Tests sowie des Testprotokolls. |
| **Nächste Schritte** | * Realisieren Dokumentieren * Automatisierte Tests * Testprotokoll durchführen |

### Tag 8: Freitag, 12.04.2024

|  |  |
| --- | --- |
| **Tätigkeiten** | * Realisieren Dokumentieren * Automatisierte Tests * Testprotokoll durchführen |
| **Reflexion** | Heute war die Hauptaufgabe die Dokumentation der Realisierung. Ich habe probiert, die Dokumentation so zu strukturieren, wie der Ablauf der Anwendung stattfindet. Mit dieser Methodik bin ich auch beim Realisieren vorgegangen, und ich fand es war der einfachste Weg dies einer dritten Person zu erklären. Das Dokumentieren der Realisation dauerte insgesammt auch viel länger als gedacht. Ich musste mir viele Gedanken machen, zu wie ich das Prozess und die neue Implementationen beschreiben kann.  Auf dem Zeitplan bin ich aber immernoch gut unterwegs, da ich für andere Schritte weniger Zeit als gedacht gebraucht habe. Ein Beispiel dafür wäre die Ausführung der automatisierten Tests, an der ich heute auch dran geschaffen habe und innerhalb einer Stunde schon alles erfolgreich ausführen konnte. Ebenfalls habe ich heute mit dem Testprotokoll angefangen, diesen konnte ich aber noch nicht ganz durcharbeiten, ist aber auch so auf dem Plan geplannt. |
| **Nächste Schritte** | * Kontrollieren Dokumentieren * Automatisierte Tests * Testprotokoll durchführen |

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

*Kommunikation zum Backend*

Die Verbdindung des Frontends zum Backend erfolgt über ein Client-API-Package. Dieser wird automatisch von OpenAPI basierend auf dem Backend generiert. Somit wird die typensichere Kommunizierung zwischen dem OVWEB Frontend und OVWEB Backend sichergestellt.

*Server Actions*

Für Aufrufe auf der OVWEB Backend API werden im Frontend Server Actions eingesetzt. Dieses lässt Server Seitiges Programmieren zu.

Storybook

Komponente und Seiten werden im Storybook eingetragen, damit diese ohne Starten der Applikation und somit von einem Non-Developer betrachtet werden können.

**A white background with red and blue text

Description automatically generatedOffer Dashboard**

Die Offerten Dashboard bietet eine Übersicht aller existenten Offerten dar. Im Rahmen dieser IPA wird diese Seite nicht angepasst, ist aber wichtig, weil man durch diese auf die Detail Seite und auf die Erstellungsseite kommt. Um eine Offerte zu erstellen bedient man den „Neue Versicherung“ Knopf. Um eine bestehende Offerte zu berechnen und um diese in Detail zu betrachten, clickt man den View Details Text.

**A white paper with red and blue lines

Description automatically generatedOffer Creation**

Die Erstellungsseite erlaubt das Erstellen einer Offerte. Das Formular beihaltet frontend-seitiges Fehlerhandling mittels Zod. Beim anclicken des Speichern Knopfes wird die Server Action „create-offer“ aufgerufen, wo die Offerte im Backend gespeichert wird, egal ob diese die Business-Rules verletzt oder nicht.

**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

#### 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. 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 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.

Bschreiben was nicht getestet wurde, performance test weil benutzer sind berater und nicht endkunden, und davon gibts wenige also wenig systemleistungsgrenzen, mobile user experience cause replaceemnt for desktop app and only used in desktop

Enterprise software

Little users as berater

#### 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 automatisierten 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 für die Entscheidungsphase definiert. Es wird entschieden, welches der zwei Varianten der Offer Details Seite im Kapitel 9.2 implementiert werden sollte. Da die Benutzerfreundlichkeit und Ästhetik Teil der IPA Kriterien sind werden sie hier gleich gewichtet. Ebenfalls steht die Komplexität auch auf dem gleichen Niveau, da das Design benutzerfreundlich und ästhetisch klar sein muss, wird die Implementation der Seite auch Zeit in anspruch nehmen, egal für welche Variante entschieden wird.

|  |  |
| --- | --- |
| **Kriterium** | **Begründung** |
| **Benutzerfreundlichkeit** | Die Benutzbarkeit steht im Vordergrund weil es wichtig ist, dass die Prämien zum Vergleich im Vordergrund stehen. |
| **Komplexität** | Wenn die Implementation einfacher ist, bleibt mehr Zeit für andere Aufgaben. |
| **Ästhetik** | Die Ästhetik ist 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.

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

**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, und die zweite weil wir Tabellen vom Komponenten Library Shadcn UI benutzen, welche aus Erfahrung recht schwierig sein können umzusetzen.

Ästhetik:

Die Ästhetik der ersten Variante ist viel schöner und moderner. Generali setzt ebenfalls auf Ihre Website generali.ch mehr Kacheln ein als Tabellen.

## 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 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 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, *vs2* und *vs3* 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 Variabeln vs2 und vs3 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.

Zuletzt beinhaltet Error das 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.  
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.

**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

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

Für die mapToFosoftRequest und addCustomerInfoToRequest Methoden wurde dies bewusst nicht getan, weil die Methoden statisch sind und somit nicht in Grafana auftauchen würden.

**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, der 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));

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 ein Error Code so wie das vom ANGEBOTSoftware generierte Fehler zurückgewiesen, 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.

*checkOfferValidity*

Diese Methode wird für die Validierung der Offerten verantwortlich sein.

Zuerst wird geloggt. Dabei loggen wir 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 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 durchlauft werden und somit für jede Versicherungssumme eine Validierung gemacht werden.  
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.

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();

In dem for Loop wird zuerst gecheckt, ob die Versicherungssumme null ist. Dies wird gemacht, weil es sein kann, dass ein Berater ein Offerten-Bundle mit nur zwei Offerten erstellt. In diesem Falle wäre die dritte Versicherungssumme somit null und es sollte nichts validiert oder berechnet werden.

Anschliessend wird ein fosoftRequest erstellt. Hier werden die vorhandenend Daten so angepasst, dass Sie der Struktur entsprechen, welche für den Endpoint D2 der ANGEBOTSoftware notwendig ist, siehe Kapitel 8.3.3. Die addCustomerInfoToRequest

Methode wird neu erstellt, diese ist notwendig, weil die Kundendaten auf dem FosoftRequest gemappt werden müssen.

Nachdem der fosoftRequest erstellt wurde, wird dieser mittels 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 ein Fehler beinhalten kann, wird im folgenden Code gecheckt, ob die Antwort so 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 dem response herausgeholt und mittels eines neues ProductCalculation Objekts zu der Liste der responses hinzugefügt.

Dies wird gemacht, damit wir dem Frontend auch übergeben können, bei der Berechnung welcher Versicherungssumme das Fehler geschehen ist, ansonnsten könnte man den Response auch direkt von der externen API zurückgeben, weil dieser auch der ProductCalculationResponse Struktur entspricht, genau wie die errorCalculation, welche in der vorher beschriebenen Methode zu den Responsen hinzugefügt wurde.

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 gesehen kann, dass es nicht eine Verletzung der Business-Rules war, sondern ein Fehler mit dem API Call.

*addCustomerInfoToRequst*

Diese Methode wurde beim Refactoring des Codes implementiert. Gemerkt wurde, dass die neu geschriebene Methode, mapToFoSoftRequestValidityCheck, redundant war. Sie gleichte zu sehr der mapToFosoftRequest Methode.

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 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;  
}

*calculateOffer*

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

### Frontend

#### Actions

Um das Erlauben eines Offerten-Bundles so wie die 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 Props genutzt, um die Anpassungsfähigkeit und Flexibilität des Komponenten zu gewährleisten.

Das Styling des 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 für die möglichkeit dreier Versicherungssummen implementiert. Ebenfalls wird Validierung mittels OVWEB Backend und ANGEBOTSoftware implementiert.

Zuerst musst das erste Feld von «VS» zu «VS1» umebenannt werden. Wegen der Umbenennung braucht es Änderungen im Zod Validation, im onSubmit so wie im Form angepasst werden.

Die Logik von VS1 wird dann kopiert und zwei weitere Male dupliziert. Somit entstehen zwei neue VS Felder, VS2 und VS3. Es braucht kleine Anpassungen am Styling damit jedes auf der gleiche Höhe stehen und umgefähr ein drittel des Parents Divs benutzt.

Für die Benutzerfreundlichkeit wurden im useEffect States benutzt, damit das VS2 Feld auch nur dann benutzbar ist, wenn man schon eine VS1 Summe eingefüllt hat. Das VS3 Feld wird dann bedienbar, sobald die VS2 Summe befüllt wurde.

Für das Empfangen und Anzeigen von Fehlern bei der Erstellung wie die Verletzung der Business-Rules wird das 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 Detail Seite 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.

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

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 Values 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 die neue Methoden der LifeInsuranceCalculation

#### Frontend Jest Tests

Im Frontend wurde die neue Komponente ProductCalculationDisplay getestet.

Es wurden alle neue Komponenten im Storybook des Projektes erfasst, dabei wurden Positiv so wie negativ Szeniarien integriert wie in folgende Beispiele. Den Rest der Tests finden Sie im Anhang mit dem restlichen Code.

A screenshot of a phone

Description automatically generatedA screenshot of a computer

Description automatically generated

## 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