TailoringExpert Architekturdokumentation

Inhaltsverzeichnis

Versionshistorie . . . 5
1. Einführung und Ziele . . . 6
1.1. Aufgabenstellung . . . 6
1.1.1. Anforderungen . . . 6
1.2. Qualitätsziele . . . 6
1.3. Stakeholder . . . 6
2. Randbedingungen . . . 7
2.1. Technische Randbedingungen . . . 7
2.2. Organisatorische Randbedingungen . . . 7
2.3. Konventionen . . . 7
3. Kontextabgrenzung . . . 9
3.1. Fachlicher Kontext . . . 9
3.1.1. Anwender (Benutzer) . . . 9
3.1.2. DRD (Fremdsystem) . . . 9
3.2. Technischer Kontext . . . 9
3.2.1. Webbrowser (Fremdsystem) . . . 9
3.2.2. Teamsite (Fremdsystem) . . . 9
4. Lösungsstrategie . . . 10
4.1. Aufbau von Tailoring . . . 10
4.2. Mandantenfähigkeit . . . 10
5. Bausteinsicht . . . 12
5.1. Whitebox Gesamtsystem . . . 12
5.1.1. tailoringexpert-core . . . 13
5.1.2. tailoringexpert-tenant . . . 13
5.1.3. tailoringexpert-data-jpa . . . 13
5.1.4. tailoringexpert-openhtmltopdf . . . 13
5.1.5. tailoringexpert-poi . . . 13
5.1.6. tailoringexpert-rest . . . 13
5.1.7. tailoringexpert-vue . . . 14
5.2. Ebene 2 . . . 14
5.2.1. Whitebox *tailoringexpert-core* . . . 14
5.2.2. Whitebox *tailoringexpert-tenant* . . . 15
5.2.3. Whitebox *tailoringexpert-data-jpa* . . . 15
5.2.4. Whitebox *tailoringexpert-openhtmltopdf* . . . 17

5.2.5. Whitebox *tailoringexpert-poi* . . . 18
5.2.6. Whitebox *tailoringexpert-rest* . . . 18
5.3. Ebene 3 . . . 20
5.3.1. Whitebox *tailoringexpert-core:domain* . . . 20
5.3.2. Whitebox *tailoringexpert-core:projekt* . . . 21
5.3.3. Whitebox *tailoringexpert-core:tailoring* . . . 22
5.3.4. Whitebox *tailoringexpert-core:anforderung* . . . 22
5.3.5. Whitebox *tailoringexpert-core:screeningsheet* . . . 22
5.3.6. Whitebox *tailoringexpert-core:katalog* . . . 23
5.3.7. Whitebox *tailoringexpert-core:renderer* . . . 23
5.3.8. Whitebox *tailoringexpert-data-jpa:domain* . . . 24
5.3.9. Whitebox *tailoringexpert-data-jpa:repository* . . . 24
5.3.10. Whitebox *tailoringexpert-data-jpa:projekt* . . . 24
5.3.11. Whitebox *tailoringexpert-data-jpa:katalog* . . . 25
5.3.12. Whitebox *tailoringexpert-data-jpa:tailoring* . . . 26
5.3.13. Whitebox *tailoringexpert-data-jpa:anforderung* . . . 26
5.3.14. Whitebox *tailoringexpert-data-jpa:screeningsheet* . . . 27
5.3.15. Whitebox *tailoringexpert-openhtmltopdf:tailoring* . . . 27
5.3.16. Whitebox _tailoringexpert-openhtmltopdf:katalog . . . 28
5.3.17. Whitebox *tailoringexpert-rest:anforderung* . . . 28
5.3.18. Whitebox *tailoringexpert-rest:domain* . . . 28
5.3.19. Whitebox *tailoringexpert-rest:katalog* . . . 29
5.3.20. Whitebox *tailoringexpert-rest:projekt* . . . 29
5.3.21. Whitebox *tailoringexpert-rest:tailoring* . . . 30
5.3.22. Whitebox *tailoringexpert-rest:screeningsheet* . . . 30
5.3.23. Whitebox *tailoringexpert-tenant:katalog* . . . 31
5.3.24. Whitebox *tailoringexpert-tenant:renderer* . . . 31
5.3.25. Whitebox *tailoringexpert-tenant:screeningsheet* . . . 31
5.3.26. Whitebox *tailoringexpert-tenant:tailoring* . . . 32
5.3.27. Whitebox <_Baustein y.1_> . . . 32
6. Laufzeitsicht . . . 33
6.1. Projekt anlegen . . . 33
6.1.1. Schnittstellen Sicht . . . 33
6.1.2. Detailierte Sicht . . . 33
6.2. Tailoring Anforderungen importieren . . . 34
6.3. Projekt kopieren . . . 34
6.4. Projektphase anlegen . . . 35
6.5. Neue Anforderung hinzufügen . . . 35
7. Verteilungssicht . . . 36
7.1. Natives Deployment . . . 36
7.2. Virtualisiertes Deployment mit Docker . . . 36

7.2.1. Guests . . . 36
7.2.2. Netzwerke . . . 37
8. Querschnittliche Konzepte . . . 38
8.1. Erzeugung von Datenobjekten . . . 38
8.2. Datentypen . . . 38
8.3. Entitäten . . . 38
8.4. Datentypkonvertierung . . . 38
8.5. Autorisierung und Authentifizierung . . . 38
8.6. Dependency Injection . . . 38
8.7. Datenbankversionierung . . . 39
8.8. Testen der Architektur . . . 39
8.9. Webservices . . . 39
8.10. Mandantenproxies . . . 39
9. Entwurfsentscheidungen . . . 41
9.1. Erzeugung PA Dokumente . . . 41
9.1.1. Kontext und Problemstellung . . . 41
9.1.2. Entscheidungstreiber . . . 41
9.1.3. Betrachtete Lösungsmöglichkeiten . . . 41
9.1.4. Entscheidung . . . 41
9.1.5. Vergleich der Alternativen . . . 41
9.1.6. Links . . . 42
9.2. HTML Template Engine . . . 43
9.2.1. Kontext und Problemstellung . . . 43
9.2.2. Entscheidungstreiber . . . 43
9.2.3. Betrachtete Lösungsmöglichkeiten . . . 43
9.2.4. Entscheidung . . . 43
9.2.5. Vergleich der Alternativen . . . 43
9.2.6. Links . . . 44
9.3. Screeningsheet . . . 44
9.3.1. Kontext und Problemstellung . . . 44
9.3.2. Entscheidungstreiber . . . 44
9.3.3. Betrachtete Lösungsmöglichkeiten . . . 44
9.3.4. Entscheidung . . . 44
9.3.5. Vergleich der Alternativen . . . 44
9.3.6. Links . . . 45
10. Qualitätsanforderungen . . . 46
10.1. Qualitätsbaum . . . 46
10.2. Qualitätsszenarien . . . 46
11. Risiken und technische Schulden . . . 47
11.1. Risiken . . . 47
11.2. Technische Schulden . . . 47

12. Glossar . . . 48

Versionshistorie

Version	Datum	Änderung
0.0.1	26.11.2020	initale Version

1. Einführung und Ziele

1.1. Aufgabenstellung

Ziel ist die Entwicklung einer Webanwendung für die Erstellung von getailorten Anforderungskatalogen. Anforderungskataloge sollen dabei phasenbezogen erstellt werden können

1.1.1. Anforderungen

1.2. Qualitätsziele

Nr	Ziel	Motivation und Erläuterung
1	Technologie Neutralität	Eine zu starke Abhängigkeit des fachlichen Kerns zu Libraries ist zu vermeiden. Der fachliche Kern soll stabil sein.
2	Testbarkeit	Da das Programm eine Neuentwicklung eines bestehenden Systems ist, muss einer Vergleichbarkeit mit den Tailoringergebnissen der aktuellen Umsetzung möglich sein

1.3. Stakeholder

Rolle	Kontakt	Erwartungshaltung
Auftraggeber		*Das Programm ermöglicht und erleichert die Erstellung von Projektanforderungskatalogen*
Auftragnehmer		*Vom Auftraggeber getailorte Anforderungskataloge in maschinenverarbeitbarer Form*

2. Randbedingungen

Beim Lösungsentwurf waren zu Beginn verschiedene Randbedingungen zu beachten, sie wirken in der Lösung fort.

Dieser Abschnitt stellt sie dar und erklärt auch – wo nötig – deren Motivation.

2.1. Technische Randbedingungen

Randbedingung	Erläuterungen, Hintergrund
Implementierung in Java	Entwicklung Java SE 11. Die Engine soll auch in neueren Java-Versionen, sobald verfügbar, laufen.
Fremdsoftware frei verfügbar	Falls zur Lösung Fremdsoftware hinzugezogen wird (z.B. grafisches Frontend), sollte diese idealerweise frei verfügbar und kostenlos sein. Die Schwelle der Verwendung wird auf diese Weise niedrig gehalten.
Berücksichtigung der internen Sicherheitsrichtilinien	Die Anwendung muss so realisiert werden, dass sie im DLR Umfeld eingesetzt werden darf
Berechtigungverwaltung über Comet	Im DLR wir das Single Sign On über Comet umgesetzt. Dies ist für diese Anwendung ebenfalls erforderlich

2.2. Organisatorische Randbedingungen

Randbedingung	Erläuterungen, Hintergrund
Vorgehensmodell	
Konfigurations- und Versionsverwaltung	Die Versionsverwaltung ist mit git und Feature Branches umzusetzen
Testwerkzeuge und -prozesse	• JUnit 5 im Annotationsstil sowohl für inhaltliche Richtigkeit als auch für Integrationstests. • Akzeptanztest sind mit Cucumber umzusetzen. • Für Smoketests soll JMeter eingesetzt werden.

2.3. Konventionen

Konvention	Erläuterungen, Hintergrund
Kodierrichtlinien für Java	• Java Coding Conventions von Sun/Oracle, geprüft mit Hilfe von Findbugs • Es soll vermehrt mit *Optional* gearbeitet werden. • In RestControllern soll bei *Optional.empty* eine *404* zurück gegeben werden.
Dokumentation	• Die Dokumentation ist im Code als auch in den Dokumenten in Deutsch zu erstellen • Verwendung deutscher Bezeichner für Klassen, Methoden etc. im Java-Quelltext (es sei denn, die Java-Kodierrichtlinien stehen dem im Wege oder Präfixe sind in englischer Sprache gebräuchlich) • Für jede Schnittstellen Methode ist gültiges javadoc zu erstellen. *Webservice Schnittstellenn sind vollständig mit OpenApi Annotationen zu dokumentieren.
Architekturdokumentation	• Die Architekturdokumentation ist unter *$PROJEKT_HOME/src/site/arc42* zu erstellen • Terminologie und Gliederung nach dem deutschen arc42-Template • UML Diagramme sind mittels plantuml unterhalb von *$PROJEKT_HOME/src/site/arc42/plantuml* zu erstellen • Alle in der Dokumentation zu verwendenden Bilder sind unterhalb *$PROJEKT_HOME/src/site/arc42/images* abzulegen • Architekturentscheidungen sind als Architecture Decision Records (ADR) als einzelne Datei pro Entscheidung unterhalb von *$PROJEKT_HOME/src/site/arc42/images/09_design_decisions* fortlaufend zu dokumentieren und in *$PROJEKT_HOME/src/site/arc42/src/09_design_decisions* zu referenzieren. ◦ Ein einmal angelegter ADR darf nicht gelöscht werden. Im Fall einer gewollten Löschung ist dies als Status kenntlich zu machen

3. Kontextabgrenzung

3.1. Fachlicher Kontext

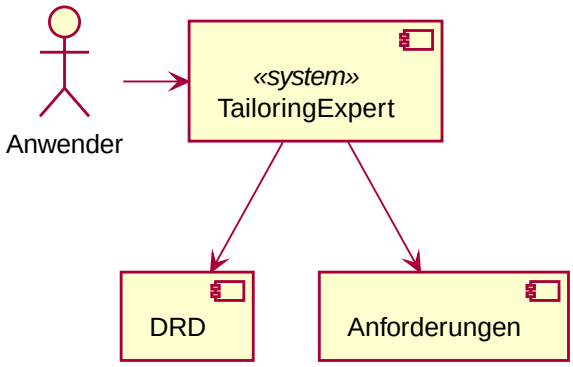

Abbildung 1. Fachlicher Kontext

3.1.1. Anwender (Benutzer)

Mitarbeiter, der projekt- und phasenspezfische Anforderungskataloge erstellt

3.1.2. DRD (Fremdsystem)

DRD sind zu verwendende oder Beispielvorlagen für den Auftragnehmer für die Erstellung der durch das Tailoring angeforderten Dokumente.

3.2. Technischer Kontext

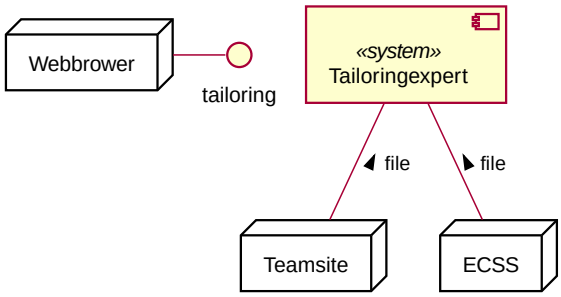

Abbildung 2. Technischer Kontext

3.2.1. Webbrowser (Fremdsystem)

Die Kommunikation zwischen den Anwendern und dem System erfolgt über einen Webbrowser.

3.2.2. Teamsite (Fremdsystem)

Die Dokumentenvorlagen werden auf einer katalogversionspezifischen Teamsite zur Verfügung gestellt. Es handelt sich hierbei in der Regel im HTML Dateien

4. Lösungsstrategie

Dieser Abschnitt enthält einen stark verdichteten Architekturüberblick. Eine Gegenüberstellung der wichtigsten Ziele und Lösungsansätze.

Die folgende Tabelle stellt die Qualitätsziele von Tailoring (siehe Abschnitt 1.2) passenden Architekturansätzen gegenüber, und erleichtert so einen Einstieg in die Lösung.

Qualitätsziel	**Dem zuträgliche Ansätze in der Architektur**
Technologie Neutralität	• Implementierung gegen Standard Schnittstelle, z.B. JPA • Umsetzung hexagonale Architektur
Testbarkeit	• Bereistellung von Cucumber für die Erstellung von Abnahme-/Akzeptanztests • Smoketest mittels gespeicherten JMeter Requests

4.1. Aufbau von Tailoring

Tailoring ist als Java Spring Anwendung unter berücksichtigung der hexagonalen Architektur realisiert.

Es zerfällt grob in folgende Teile:

- eine Implementierung des fachlichen Kerns
- einem Modul für die Datenzugriffsschicht
- Module für die Erstellung der Ausgabedokument
- einem Modul für den Import von Tailoring Anforderungen
- einem Modul mit Spring RestController für die Anbindung durch ein Webfrontend
- ein Webfrontend

Diese Zerlegung ermöglicht es, den fachlichen Kern technologieneutral zu implementieren. Alle Teile sind durch Schnittstellen abstrahiert, die Implementierungen werden per Java Config Dependency Injection zusammengesteckt.

4.2. Mandantenfähigkeit

TailoringExpert ist als mandantenfähige Anwendung zu realisieren.

Dafür soll für jede mandantenspezfische Funktionalität eine Schnittstelle definiert werden. Jede Implementierung einer Schnittstelle ist konsistent über alle Implementierungen eines Mandanten zu annotieren. Der Name der Annotation entspricht dabei dem Mandantennamen.

Die Anwendung selbst implementiert die primäre Schnittstellenimplementierung und ist für das Einbinden der Mandantenimplementierungen zuständig. Diese primäre Schnittstellenimplementierung ist dabei nichts weiter als eine Map mit dem Mapping des Mandanten(namen) zu dessen Implementierung. Der Zugriff auf die spezifische Mandantenimplementierung erfolgt mittels des Mandantenschlüssels über die primäre Schnittstelle(nmap).

Aus diesem Grund ist die Mandantenkennung über eine ThreadContext notwendig. Deshalb muss jeder Client die Mandantenkennung als X-Tenant Header Attribut senden. Die Anwendung liest dieses Attribut als erstes in einem Filter aus.

5. Bausteinsicht

Dieser Abschnitt beschreibt die Zerlegung von Tailoring in Module.

5.1. Whitebox Gesamtsystem

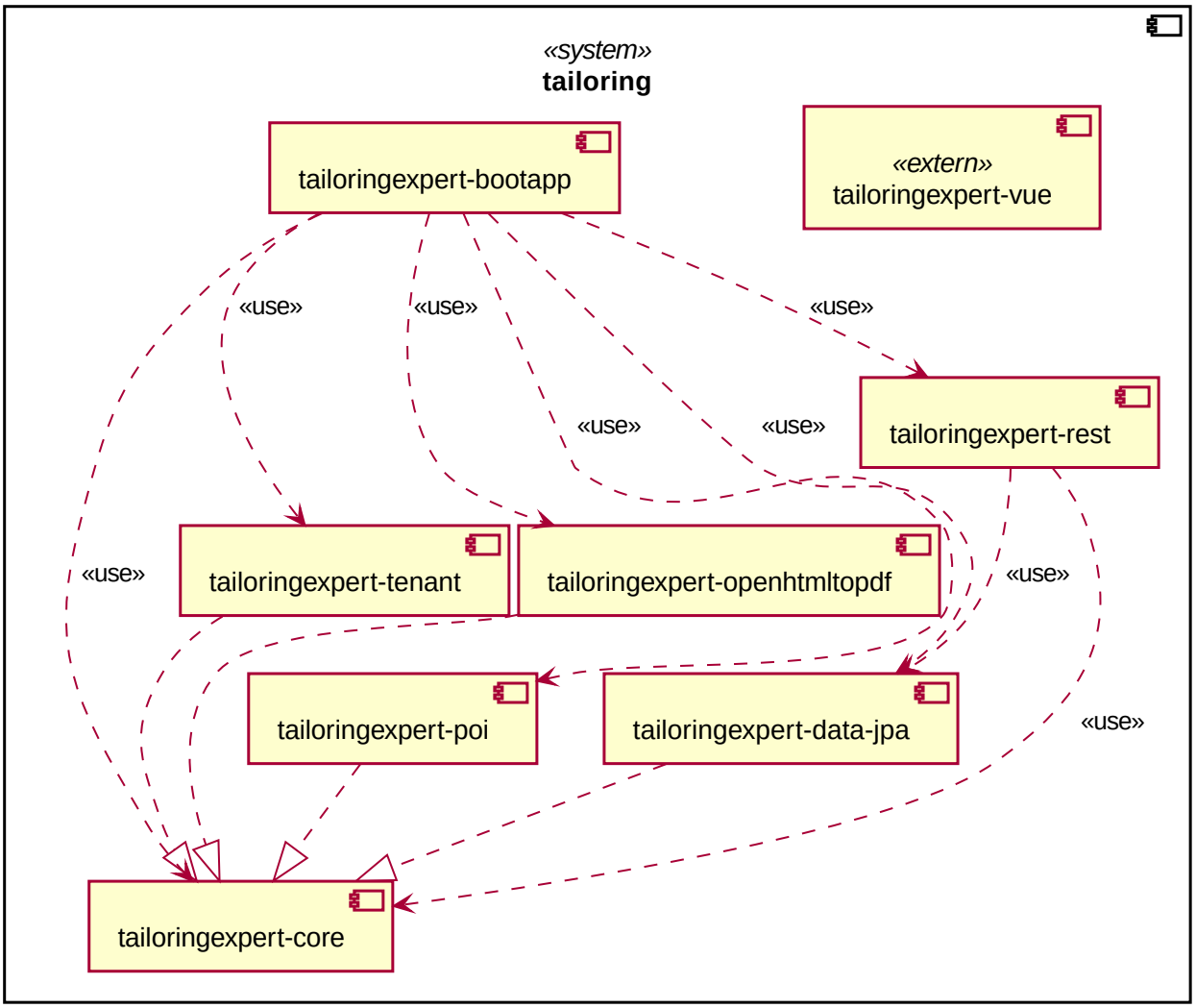

Abbildung 3. Whitebox Gesamtsystem

Begründung

Die Zerlegung in Module erfolgt nach Aspekten den hexagonalen Architektur.

Enthaltene Bausteine

Name	Verantwortung
tailoringexpert-core	*Fachlicher Kern des Systems*
tailoringexpert-tenant	*Implementierung der Mandantenproxies*
tailoringexpert-data-jpa	*Datenzugriffsmodul*
tailoringexpert-openhtmltopdf	*Generierung von Ausgabedokumenten für den Auftragsnehmer*

tailoringexpert-poi	*Verarbeitung von Tailoringanforderungen sowie Ausgabe einzelner Dokumente in Excel*
tailoringexpert-rest	*Bereitstellung einer REST-Schnittstelle für das Gesamtsystem*
tailoringexpert-vue	*Frontend des Systems/Platform*

5.1.1. tailoringexpert-core

Das Modul ist der fachliche Kern des Gesamtsystems. Alle Geschäftsprozesse werden hier umgesetzt.

Für extern benötigte Dienste oder Daten stellt dieses Modul Schnittstellen zur Verfügung. Ebenso werden hier Schnittstellen für Multi-Mandantenfähigkeit definiert.

Das Modul hat weder Abhängigkeiten zu Fremd-Libraries noch zu anderen Modulen.

5.1.2. tailoringexpert-tenant

Das Modul implementiert die Proxies für den Zugriff auf mandantenspezifische Schnittstellenimplementierungen.

5.1.3. tailoringexpert-data-jpa

Das Modul implementiert die Datenzugriffsschnittstellen des *tailoring-core* Moduls.

Der Datenzugriff erfolgt über JPA mittels *Spring Data JPA*.

5.1.4. tailoringexpert-openhtmltopdf

Modul für die Erzeugung von Ausgabedokumenten für den Auftragnehmer.

Hier werden PDF Dateien mittels *openhtmltopdf* erstellt.

5.1.5. tailoringexpert-poi

Modul für Import/Export der Anforderungen eines Tailorings.

Hier werden XLSX Dateien mittels *poi* verarbeitet.

5.1.6. tailoringexpert-rest

Bereitstellung einer REST-Schnittstelle für das Gesamtsystem.

Die REST-Schnittstelle wird über *Spring WebMVC* zur Vefügung gestellt.

Die Antworten sind *HATEOAS* konform, so dass alle erlaubten Aktionen über Links der Antwort beigefügt sind.

5.1.7. tailoringexpert-vue

Frontend für den Zugriff auf das Gesamtsystem. Wird als Single-Page App implementiert und als externes System nicht weiter in der Architekturdokumentation betrachtet.

5.2. Ebene 2

5.2.1. Whitebox *tailoringexpert-core*

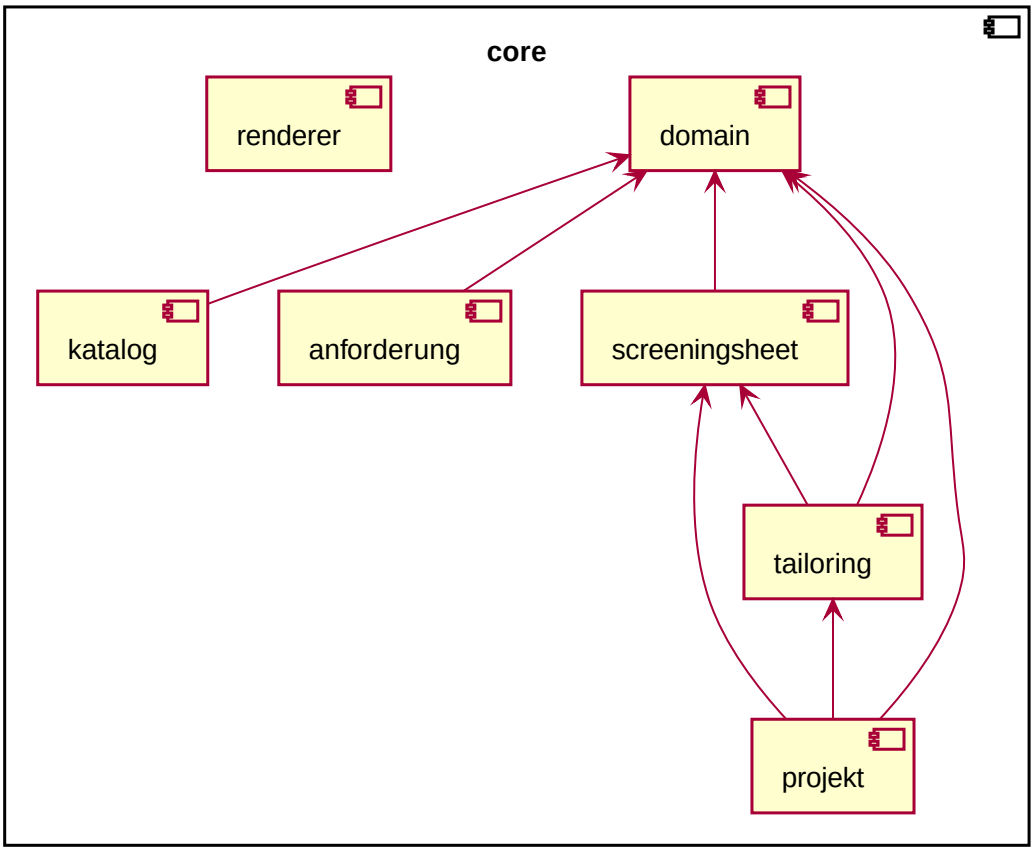

Abbildung 4. Whitebox tailoringexpert-core

Enthaltene Bausteine

Name	Verantwortung
domain	*Domänenobjekte des fachlichen Kerns*
anforderung	*Services für die Be-/Verarbeitung von Anforderungen*
katalog	*Service für den Import eines neuen Anforderungkatalogs*
projekt	*Services für die Be-/Verarbeitung von Projekten*
renderer	*Services für Bereitstellung von Rendering Engines für den Dokumentexport*
tailoring	*Services für die Be-/Verarbeitung von Tailorings*
screeningsheet	*Services für die Verarbeitung von Screeningsheets*

5.2.2. Whitebox *tailoringexpert-tenant*

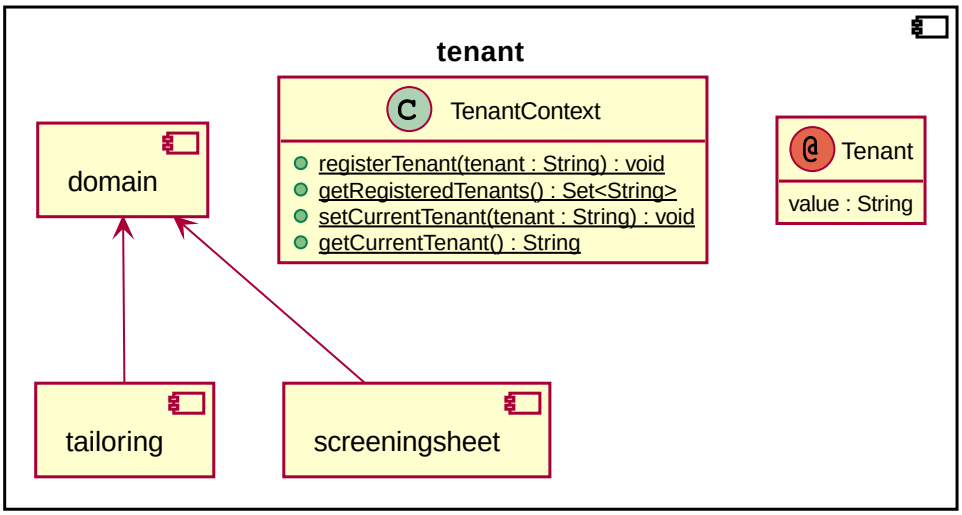

Abbildung 5. Whitebox tailoringexpert-tenant

Enthaltene Bausteine

Name	Verantwortung
tailoring	*Proxies für die mandantenspezifischen Template-Engines und Dokumentenservices*
katalog	*Proxies für die mandantenspzezifischen Gesamtkatalogerstellung*
renderer	*Proxies für die mandantenspzezifischen Selektion von Renderern für die Ausgabedokumente*
screeningsheet	*Proxies für die mandantenspzezifischen Verarbeitung von Screeningsheets*

5.2.3. Whitebox *tailoringexpert-data-jpa*

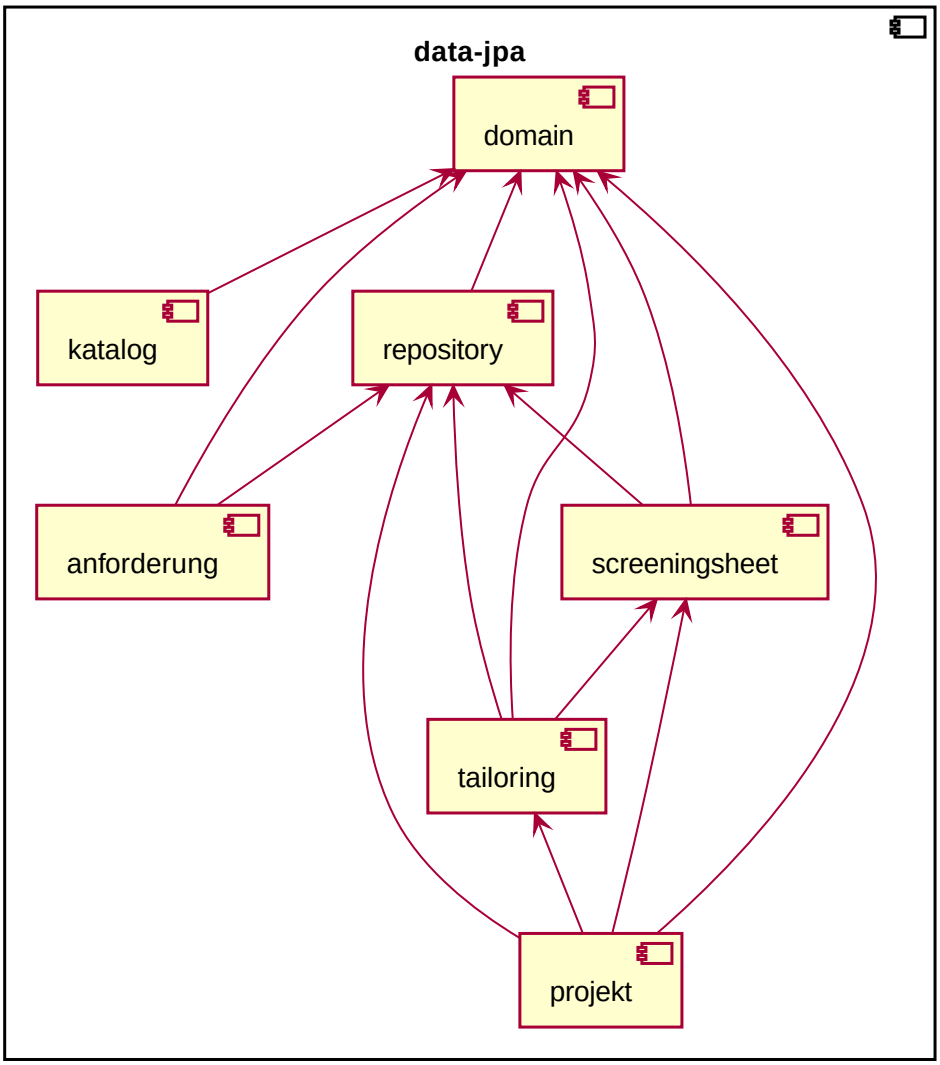

Abbildung 6. tailoringexpert-data_jpa Ebene 2

Begründung

Die Zerlegung Unterkomponenten erfolgt in Anlehung an den fachlichen Kern (*tailoring-core*). Dadurch wird ein einheitlicher Paketzugriff realisiert. Die Schnittstelle zur Datenbank ist in *Spring Data JPA Repositories* in der Komponente *repository* gekapselt.

Enthaltene Bausteine

Name	Verantwortung
domain	*Entitäten des Systems*
repository	*Datenzugriffschicht mittels **Spring Data JPA** der Entitäten*
anforderung	*Implementierungsschicht der Schnittstelle **AnforderungServiceRepository***
katalog	*Implementierungsschicht der Schnittstelle **KatalogServiceRepository***
projekt	*Implementierungsschicht der Schnittstelle **ProjektServiceRepository***

Name	Verantwortung
tailoring	*Implementierungsschicht der Schnittstelle **TailoringServiceRepository***
screeningsheet	*Implementierungsschicht der Schnittstelle **ScreeningSheetServiceRepository***

Wichtige Schnittstellen

Schnittstelle	Beschreibung
—	

5.2.4. Whitebox *tailoringexpert-openhtmltopdf*

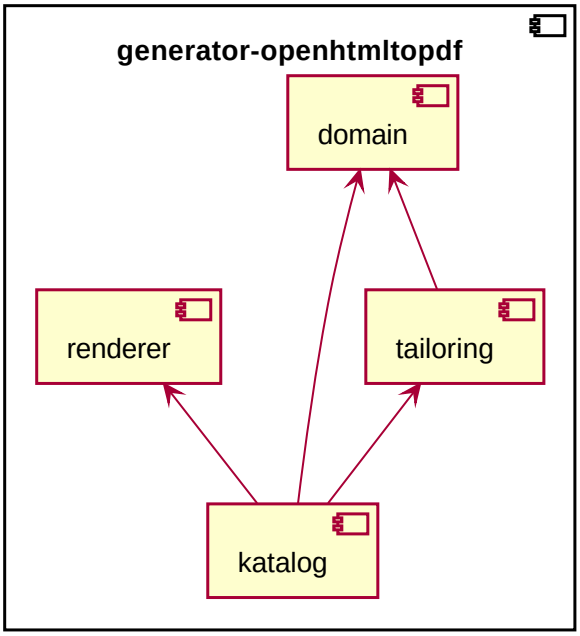

Abbildung 7. tailoringexpert-openhtmltopdf Ebene 2

Begründung

Die Zerlegung Unterkomponenten erfolgt in Anlehnung an den fachlichen Kern (*tailoringexpert-core*). Dadurch wird ein einheitlicher Paketzugriff realisiert.

Enthaltene Bausteine

Name	Verantwortung
tailoring	*Implementierungsschicht der Schnittstelle **DocumentCreator***
katalog	*Implementierungsschicht der Schnittstelle für die Gesamtkatalogerstellung **DocumentCreator***
renderer	*Implementierungsschicht der zu verwendenden Renderer Schnittstellen**

Wichtige Schnittstellen

Schnittstelle	Beschreibung
—	

5.2.5. Whitebox *tailoringexpert-poi*

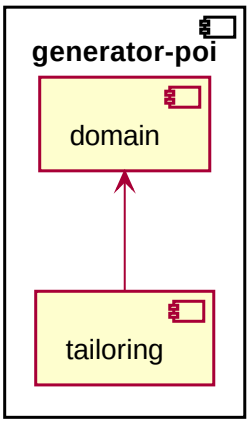

Abbildung 8. tailoringexpert-generator-poi Ebene 2

Begründung

Die Zerlegung Unterkomponenten erfolgt in Anlehung an den fachlichen Kern (*tailoring-core*). Dadurch wird ein einheitlicher Paketzugriff realisiert.

Enthaltene Bausteine

Name	Verantwortung
tailoring	*Implementierungsschicht der Schnittstelle **DocumentCreator***

Wichtige Schnittstellen

Schnittstelle	Beschreibung
—	

5.2.6. Whitebox *tailoringexpert-rest*

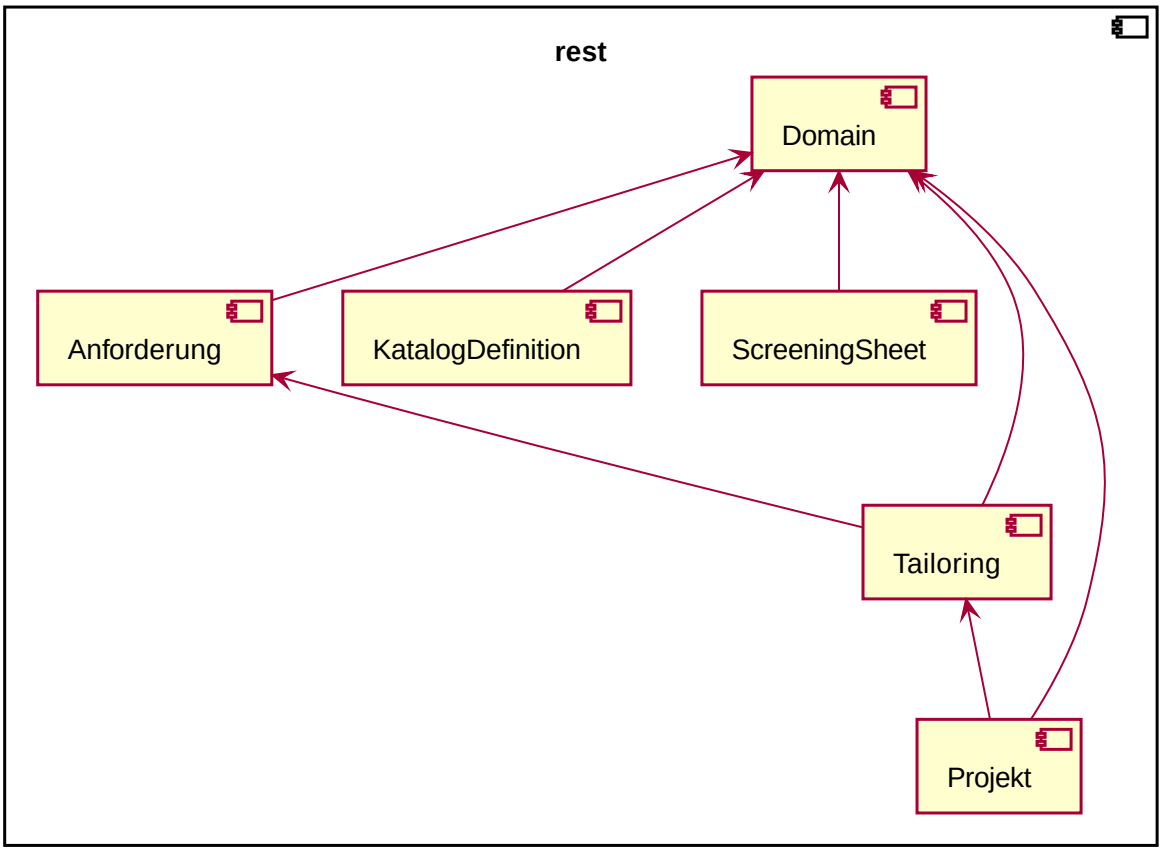

Abbildung 9. tailoringexpert-rest Ebene 2

Begründung

Die Zerlegung Unterkomponenten erfolgt in Anlehnung an den fachlichen Kern (*tailoringexpert-core*). Dadurch wird ein einheitlicher Paketzugriff realisiert. Die HATEOAS Links werden im *domain* Paket über den ResourceMapper erzeugt. Im Mapper sind entsprechende Path Elemente als Konstanten definiert. Diese sind in den Controllern für das Mapping zu verwenden.

Name	Verantwortung
anforderung	*REST-Schnittstelle für die Be-/Verarbeitung von Anforderungen*
domain	*Domänenobjekte der Komponente und ResozrceMapper für die Erzuegung der HATEOAS Links*
katalog	*REST-Schnittstelle für den Import neuer Kataloge*
projekt	*REST-Schnittstelle für die Be-/Verarbeitung von Projekten*
projektphase	*REST-Schnittstelle für die Be-/Verarbeitung von Projektphasen*
screeningsheet	*REST-Schnittstelle für die Verarbeitung von Screeningsheets*

Wichtige Schnittstellen

Schnittstelle	Beschreibung
GET /	Ermittlung der Haupt-Urls der Anforderung • Verfügbare Katalogversionen • Vorhandene Projekte • Hochladen von Screningsheets • Berechnung eines Selektionsvektors
POST <REL>	

Folgende Rels sind für die HATEOAS Links im jeweiligen Kontext definiert:

Schnittstelle	Beschreibung
SELF	Die Resource im aktuellen Kontext

Alle im Kontext einer Resource möglichen Links werden als URLs in den Serverantwortern unter *links* zur Verfügung gestellt.

5.3. Ebene 3

5.3.1. Whitebox *tailoringexpert-core:domain*

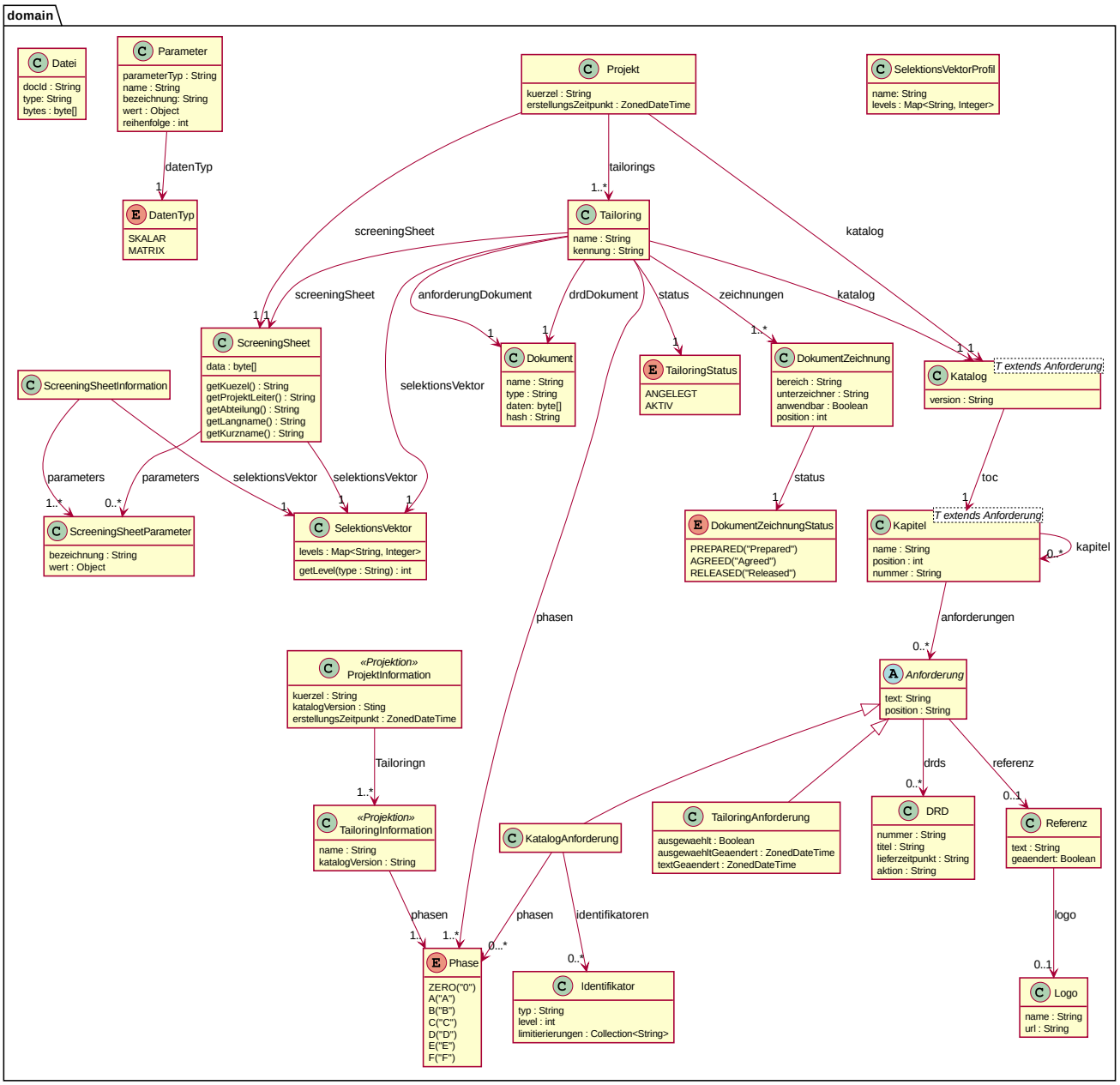

Abbildung 10. tailoringexpert-core:domain

5.3.2. Whitebox *tailoringexpert-core:projekt*

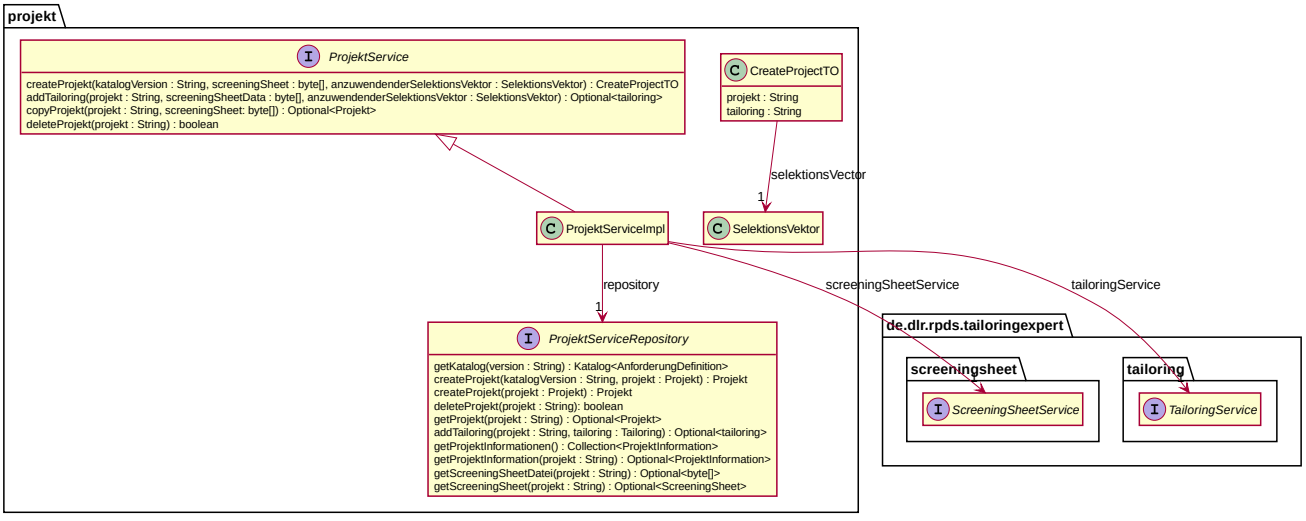

Abbildung 11. tailoringexpert-core:projekt

5.3.3. Whitebox *tailoringexpert-core:tailoring*

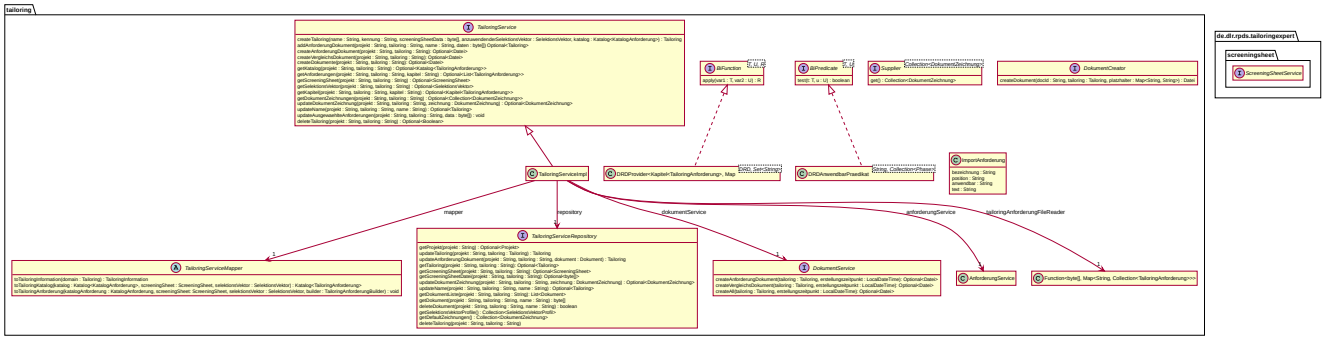

Abbildung 12. tailoringexpert-core:tailoring

Wichtige Schnittstellen

Schnittstelle	Beschreibung	Mandantspezifische Implementierung erforderlich
DokumentCreator	Interface für die Erstellung eines Dokumentes	
DokumentService	Interface für die Erstellung aller Mandantenspezifischen Dokumente.	X

5.3.4. Whitebox *tailoringexpert-core:anforderung*

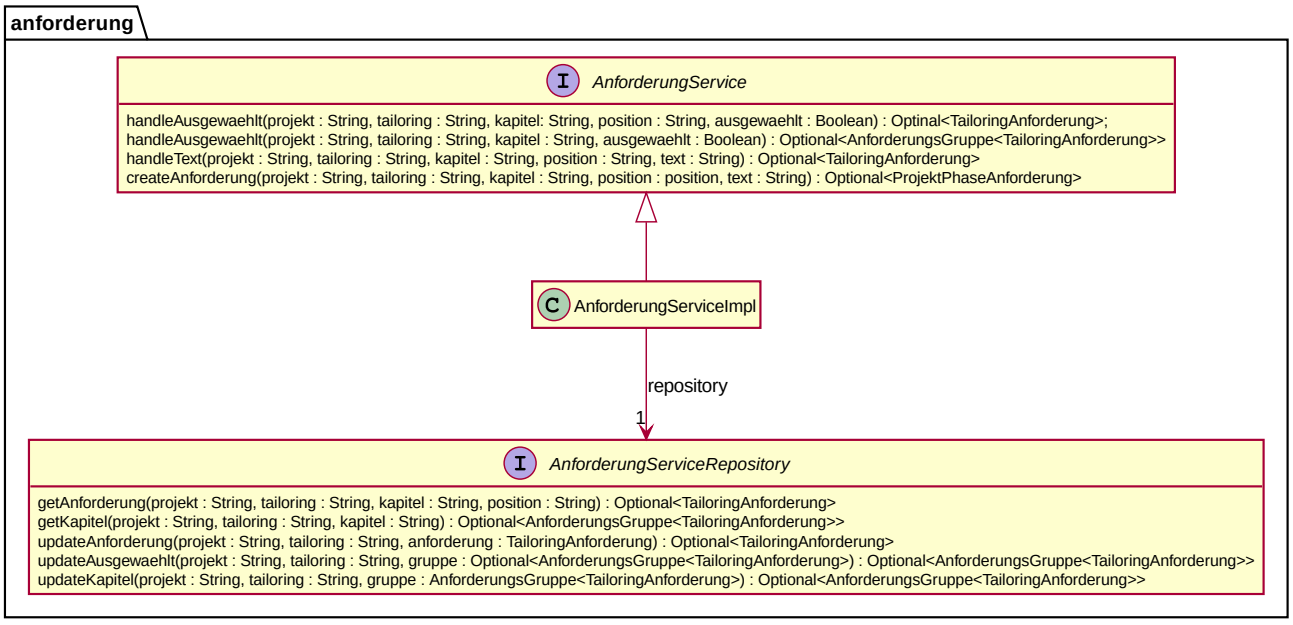

Abbildung 13. tailoringexpert-core:anforderung

5.3.5. Whitebox *tailoringexpert-core:screeningsheet*

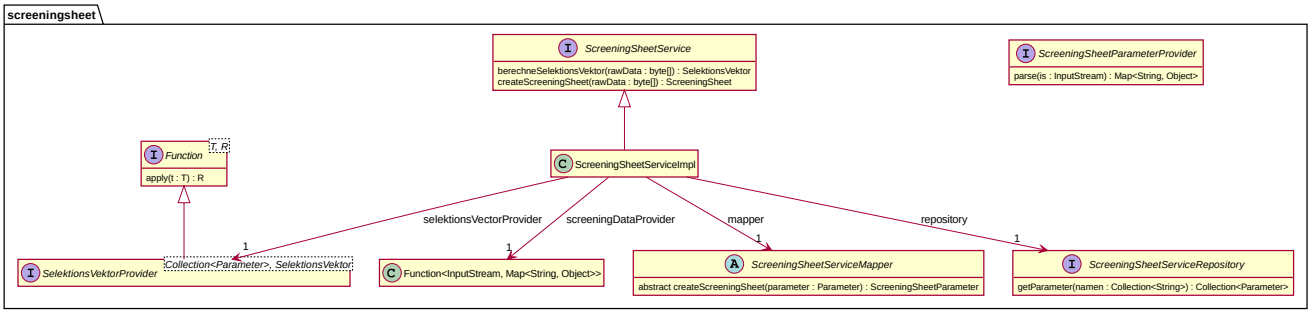

Abbildung 14. tailoringexpert-core:screeningsheet

Wichtige Schnittstellen

Schnittstelle	Beschreibung	Mandantspezifische Implementierung erforderlich
ScreeningSheetParameterProvider	Interface für die Extraktion der Mandantenspezifischen Parameter aus einem Screeningsheet.	X
SelektionsVektorProvider	Interface für die Mandantenspezifische Ermittlung des Selektionsvektors.	X

5.3.6. Whitebox *tailoringexpert-core:katalog*

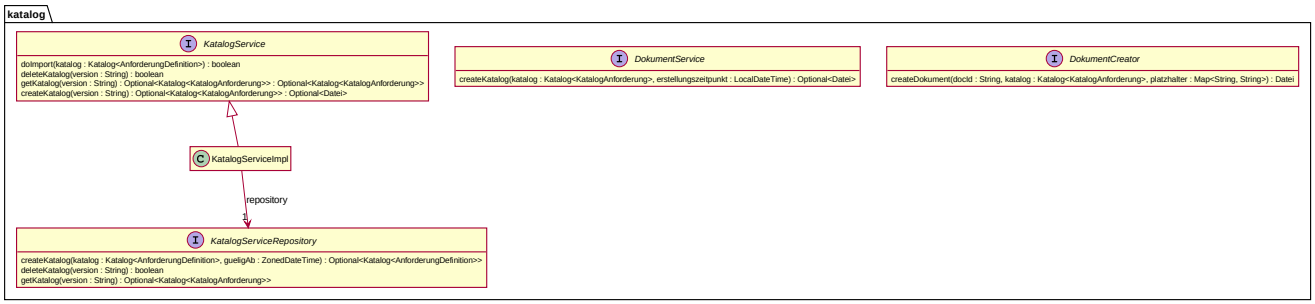

Abbildung 15. tailoringexpert-core:katalog

5.3.7. Whitebox *tailoringexpert-core:renderer*

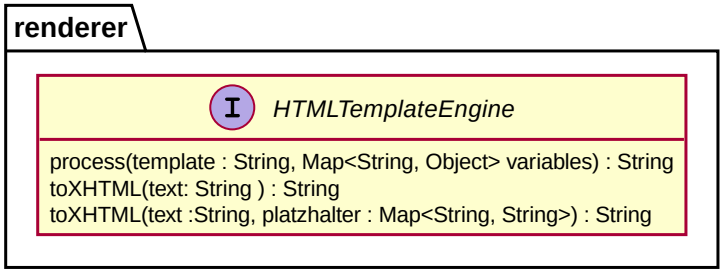

Abbildung 16. tailoringexpert-core:renderer

5.3.8. Whitebox *tailoringexpert-data-jpa:domain*

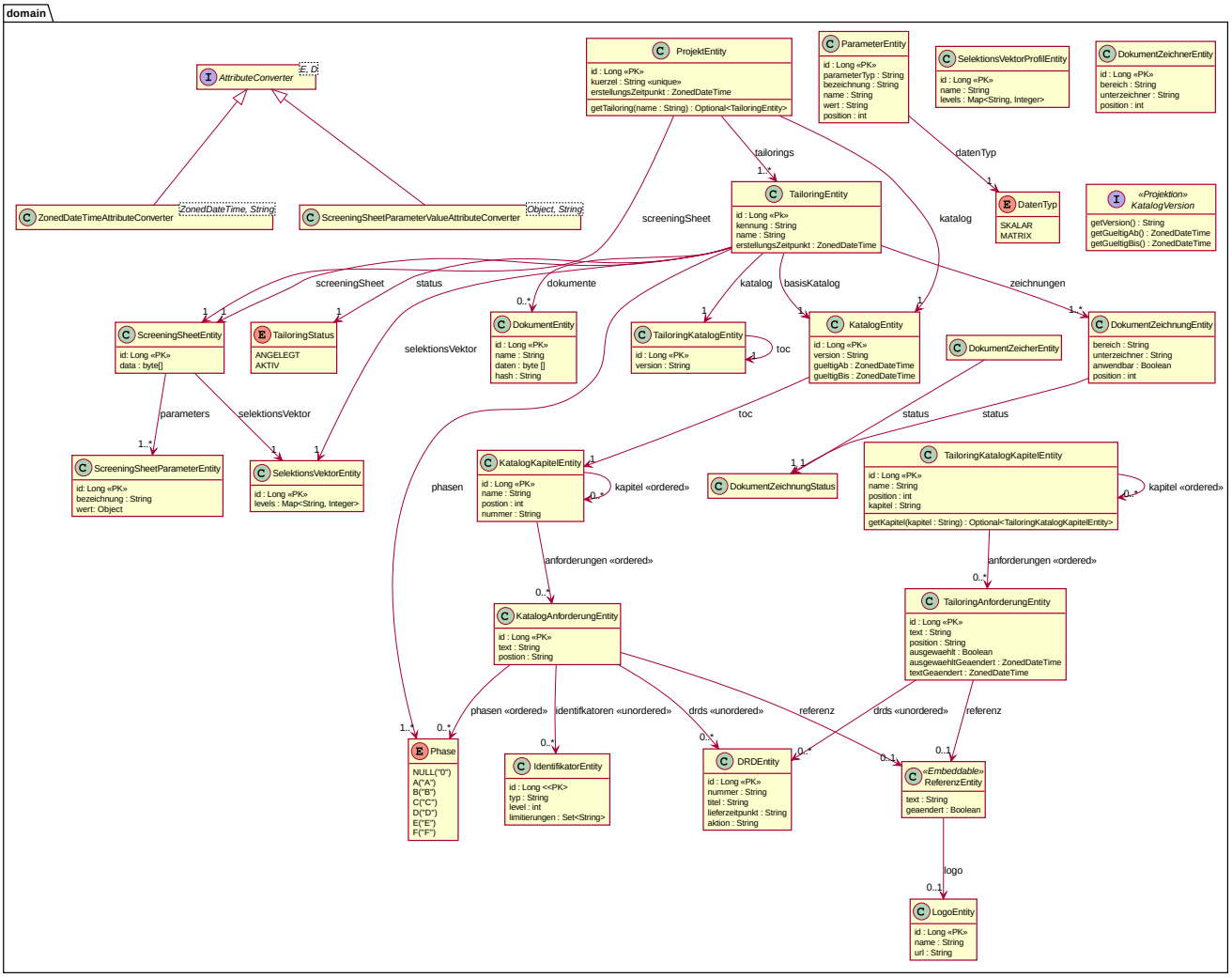

Abbildung 17. tailoringexpert-data-jpa:domain

5.3.9. Whitebox *tailoringexpert-data-jpa:repository*

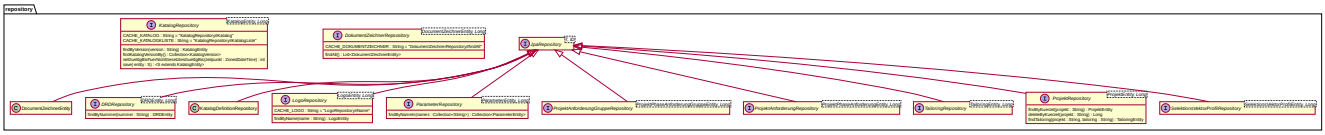

Abbildung 18. tailoringexpert-data-jpa:repository

5.3.10. Whitebox *tailoringexpert-data-jpa:projekt*

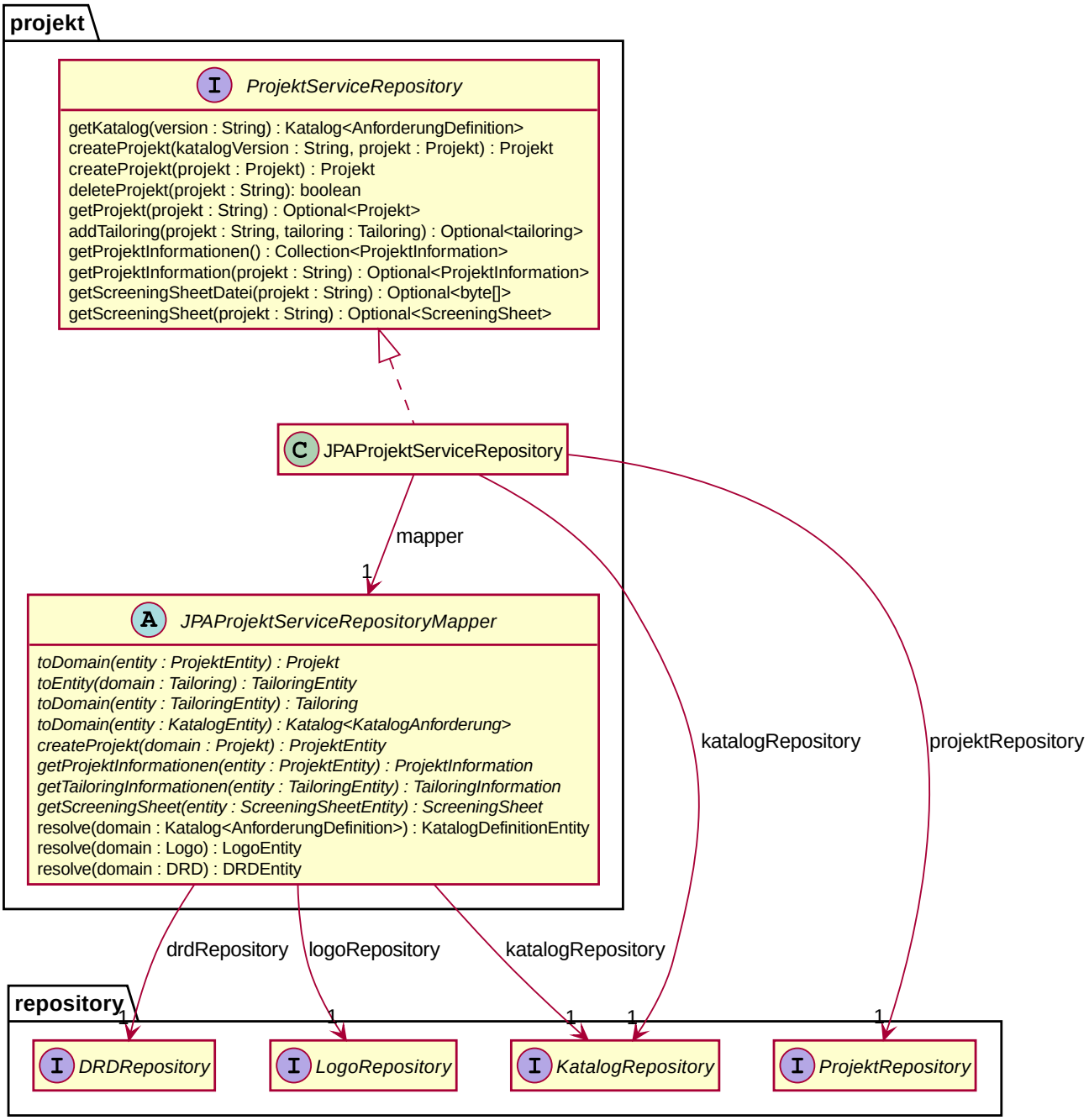

Abbildung 19. tailoringexpert-data-jpa:projekt

5.3.11. Whitebox *tailoringexpert-data-jpa:katalog*

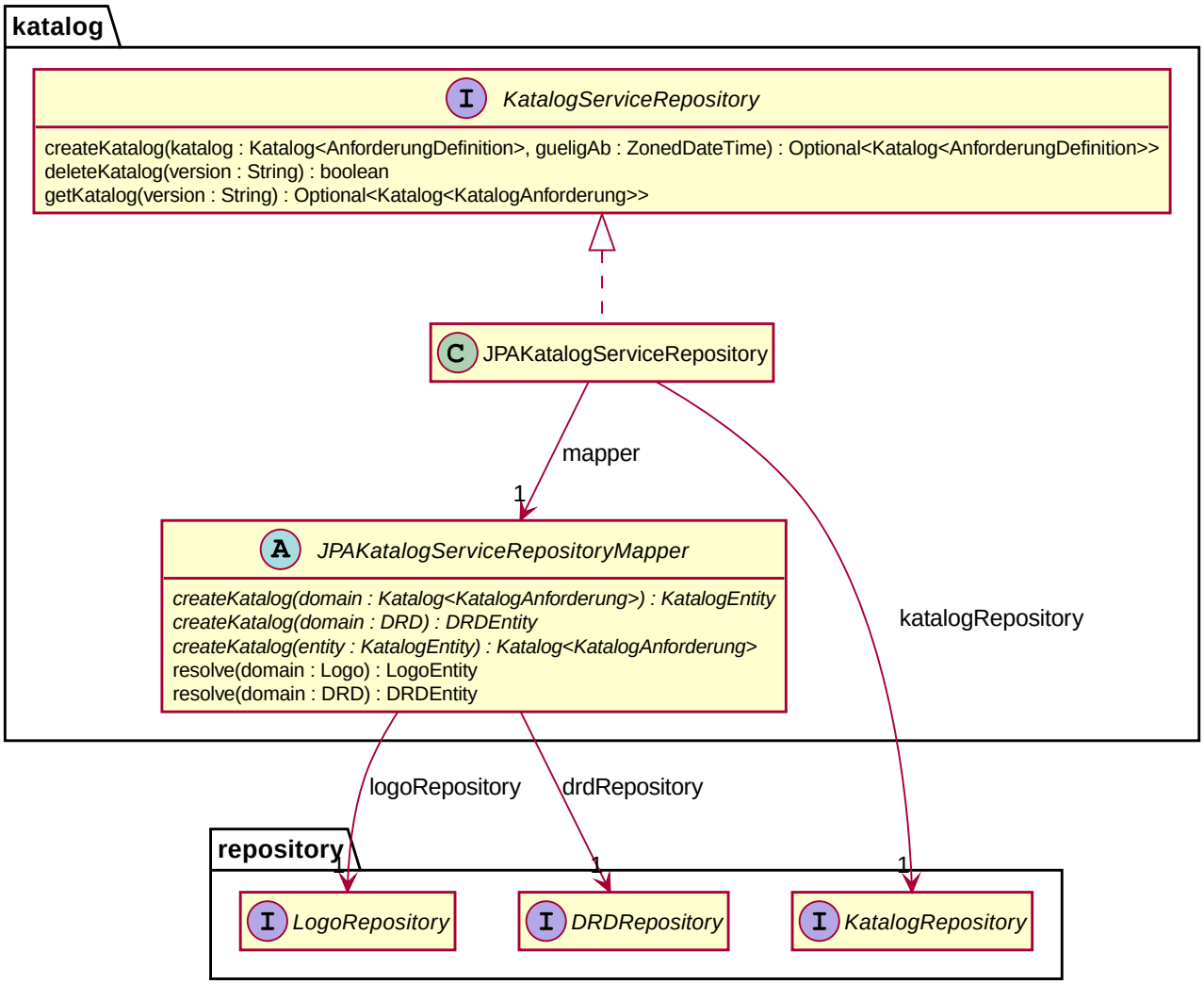

Abbildung 20. tailoringexpert-data-jpa:katalog

5.3.12. Whitebox *tailoringexpert-data-jpa:tailoring*

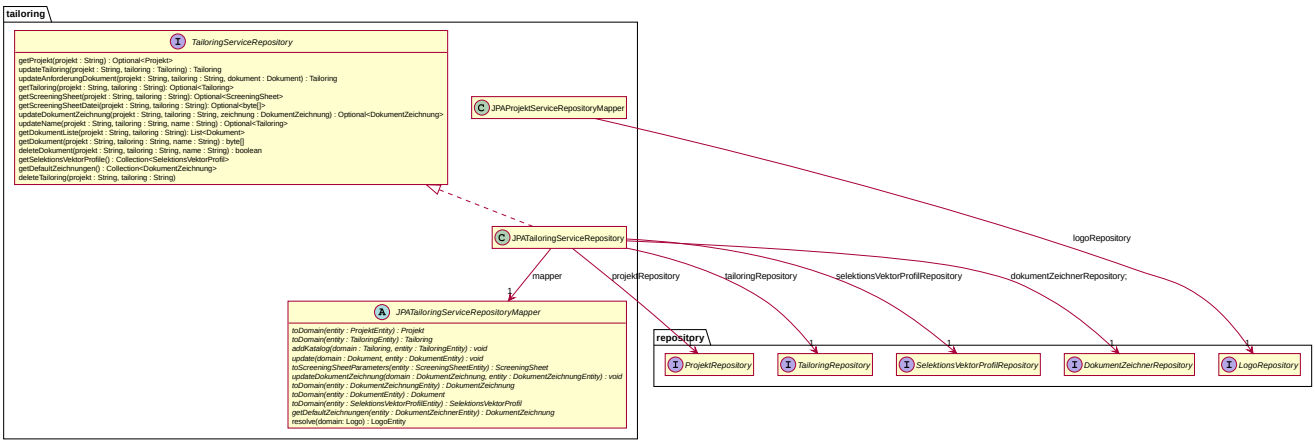

Abbildung 21. tailoringexpert-data-jpa:tailoring

5.3.13. Whitebox *tailoringexpert-data-jpa:anforderung*

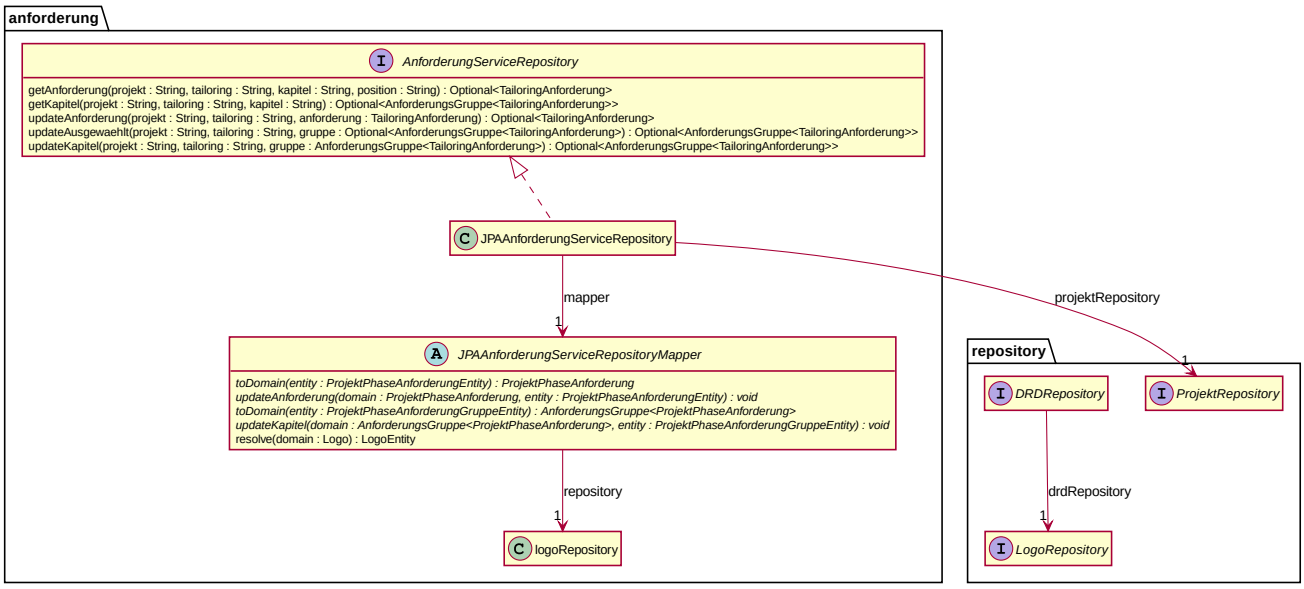

Abbildung 22. tailoringexpert-data-jpa:anforderung

5.3.14. Whitebox *tailoringexpert-data-jpa:screeningsheet*

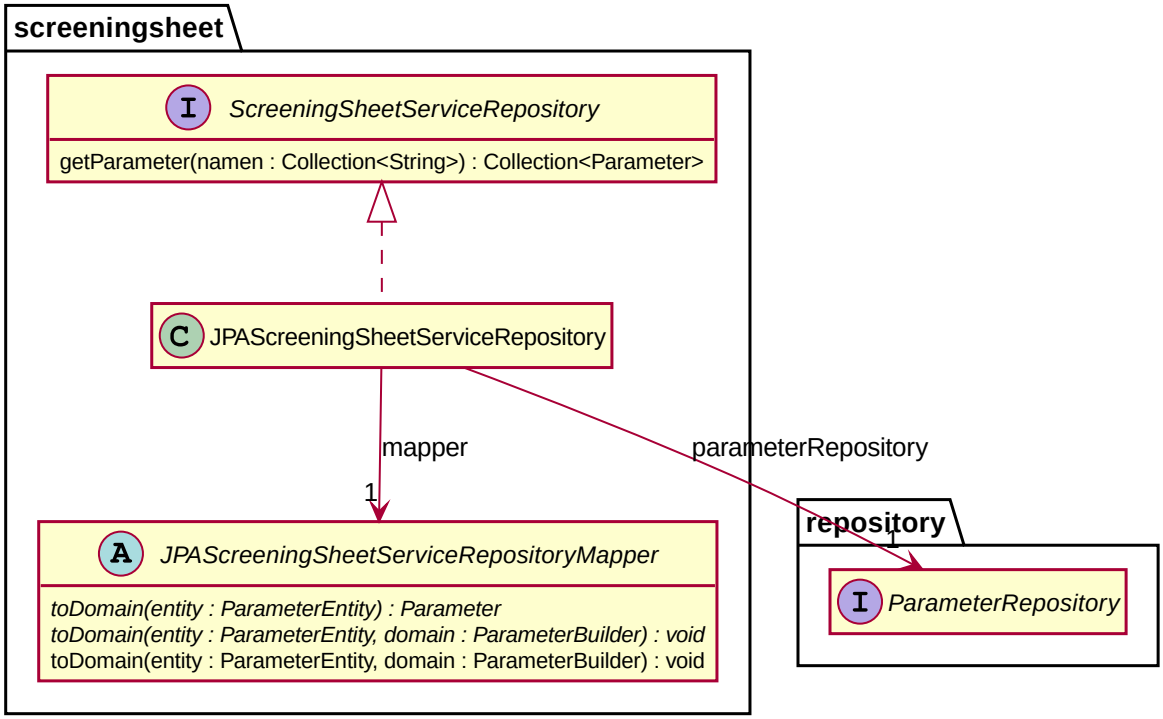

Abbildung 23. tailoringexpert-data-jpa:screeningsheet

5.3.15. Whitebox *tailoringexpert-openhtmltopdf:tailoring*

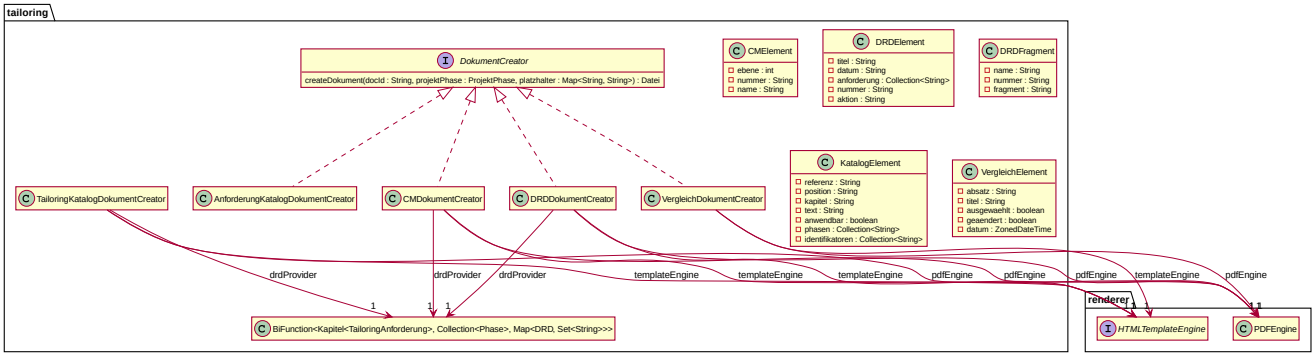

Abbildung 24. tailoringexpert-openhtmltopdf:tailoring

5.3.16. Whitebox _tailoringexpert-openhtmltopdf:katalog

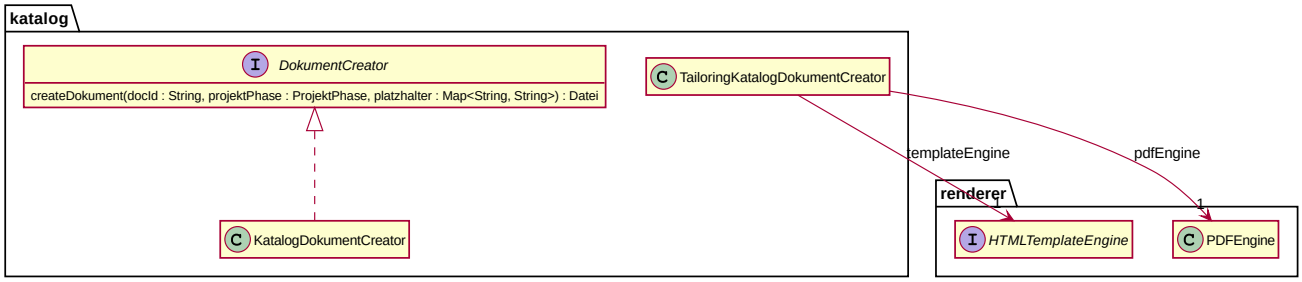

Abbildung 25. tailoringexpert-openhtmltopdf:katalog

5.3.17. Whitebox *tailoringexpert-rest:anforderung*

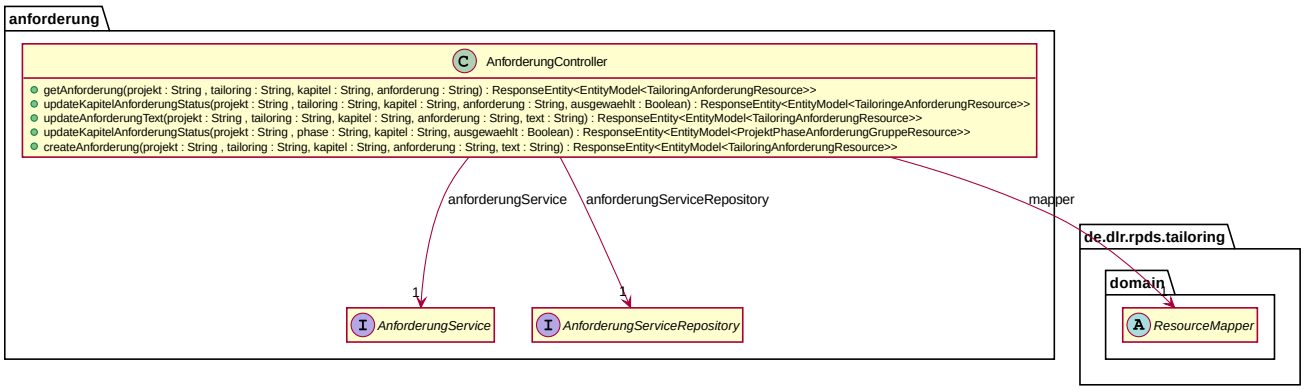

Abbildung 26. tailoringexpert-rest:anforderung

5.3.18. Whitebox *tailoringexpert-rest:domain*

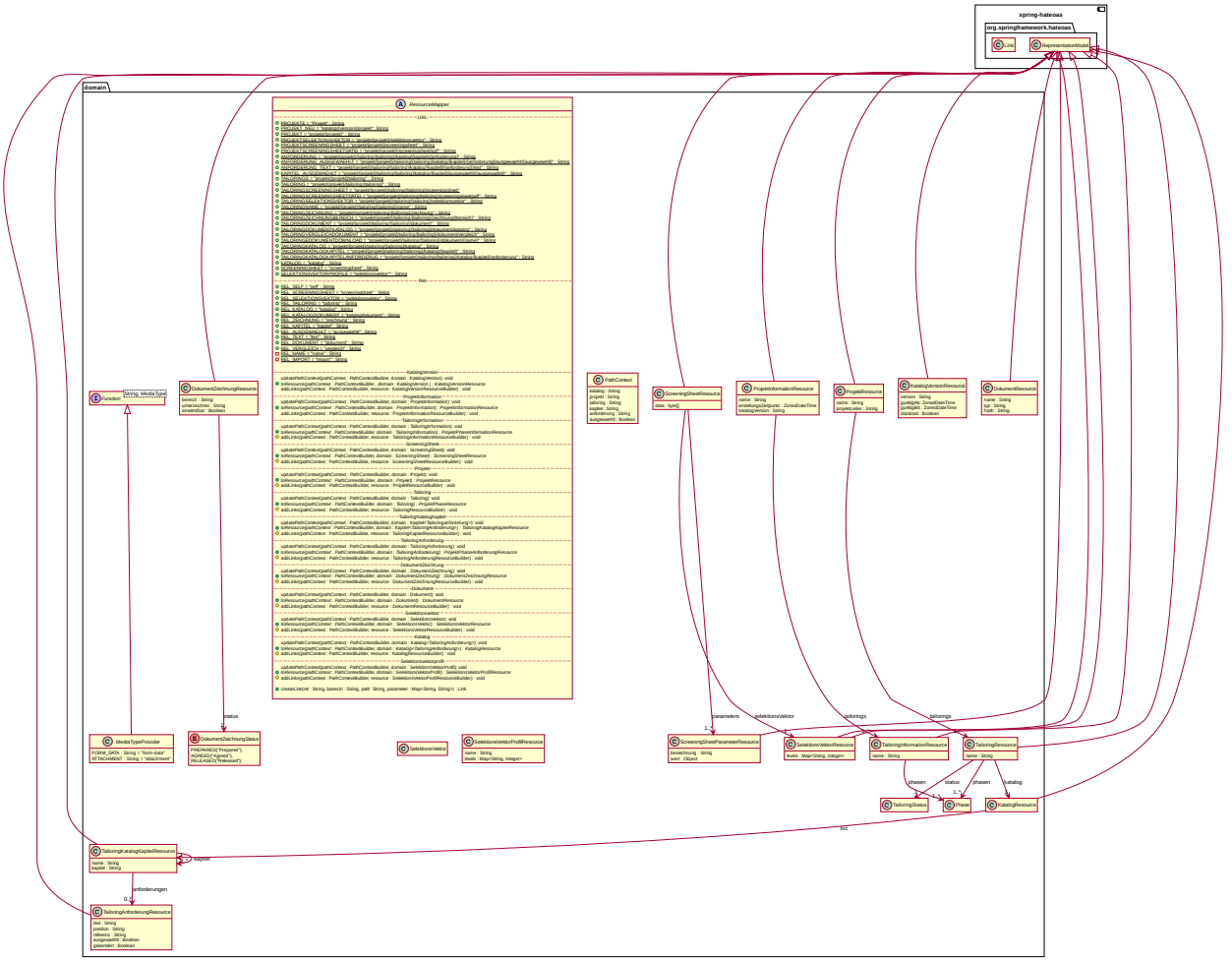

Abbildung 27. tailoringexpert-rest:domain

5.3.19. Whitebox *tailoringexpert-rest:katalog*

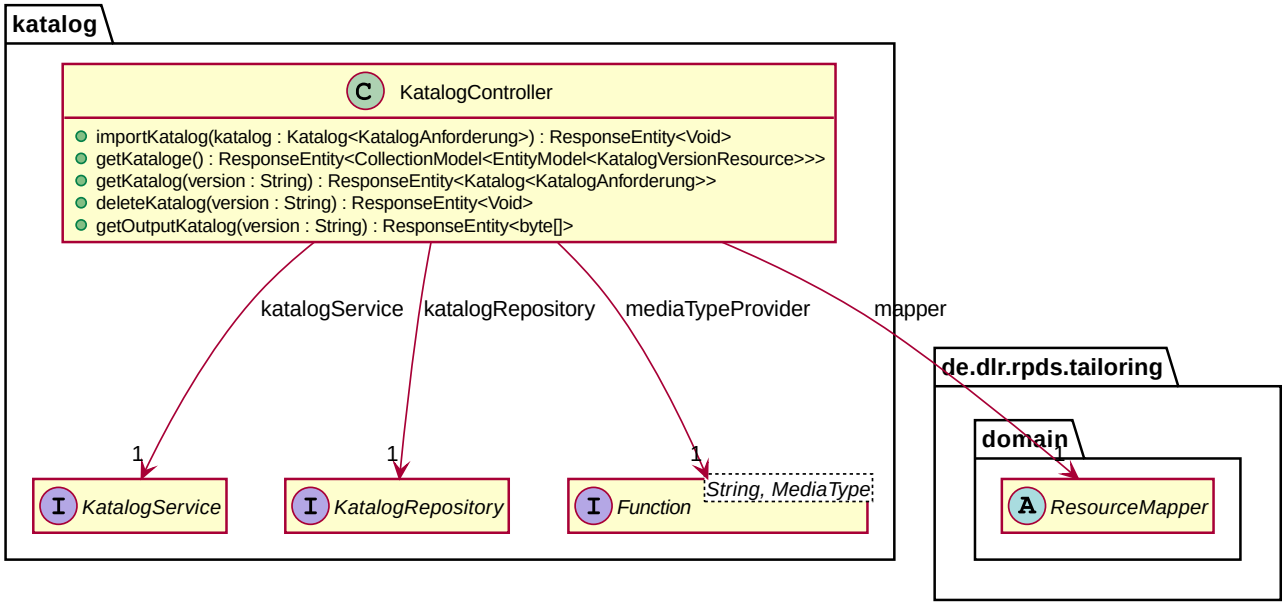

Abbildung 28. tailoringexpert-rest:katalog

5.3.20. Whitebox *tailoringexpert-rest:projekt*

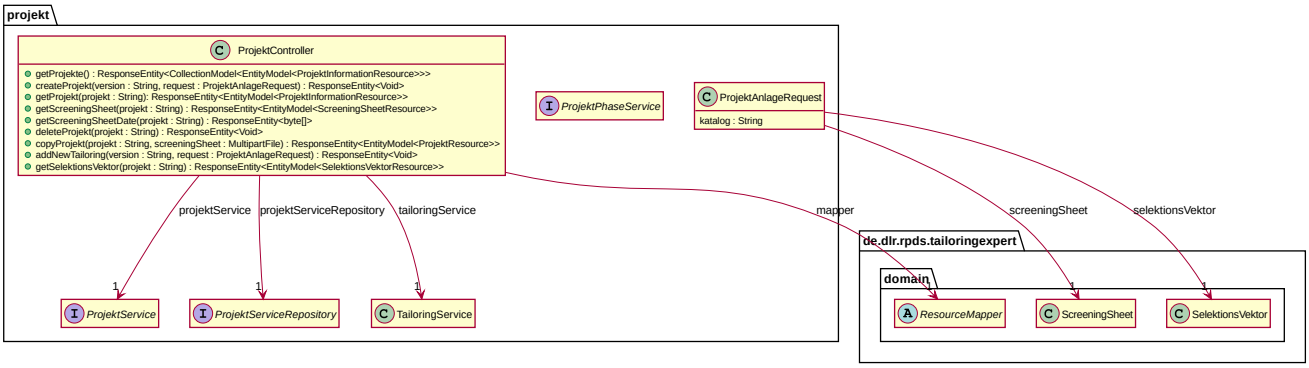

Abbildung 29. tailoringexpert-rest:projekt

5.3.21. Whitebox *tailoringexpert-rest:tailoring*

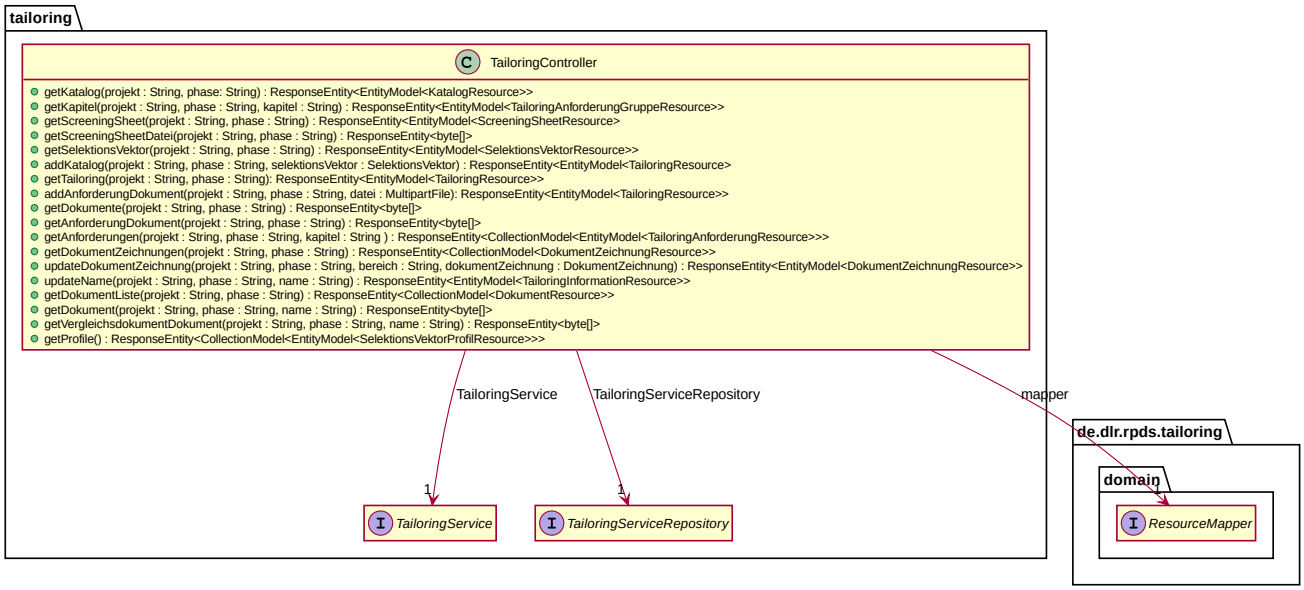

Abbildung 30. tailoringexpert-rest:tailoring

5.3.22. Whitebox *tailoringexpert-rest:screeningsheet*

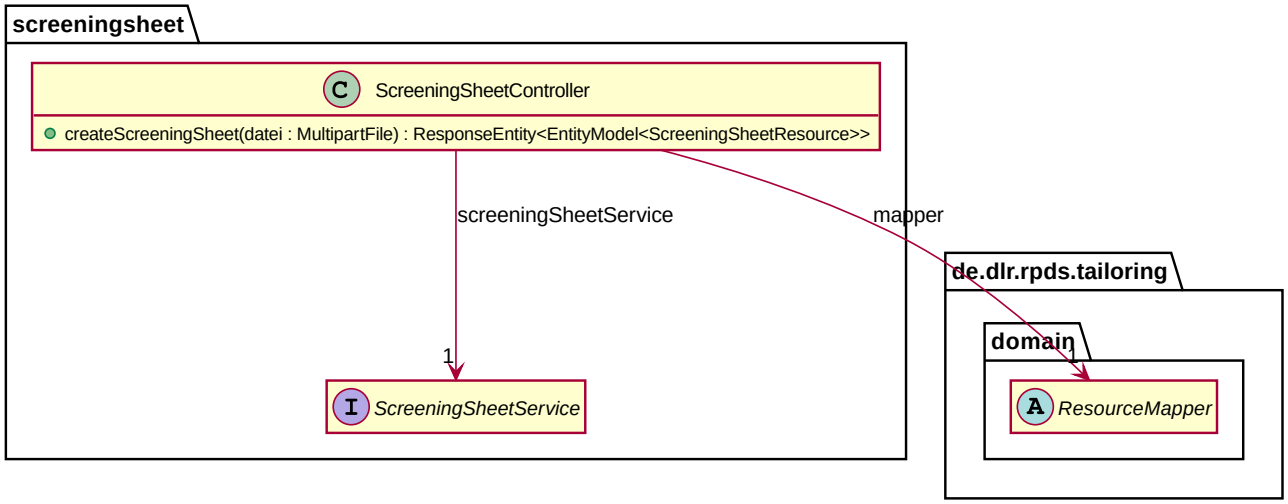

Abbildung 31. tailoringexpert-rest:screeningsheet

5.3.23. Whitebox *tailoringexpert-tenant:katalog*

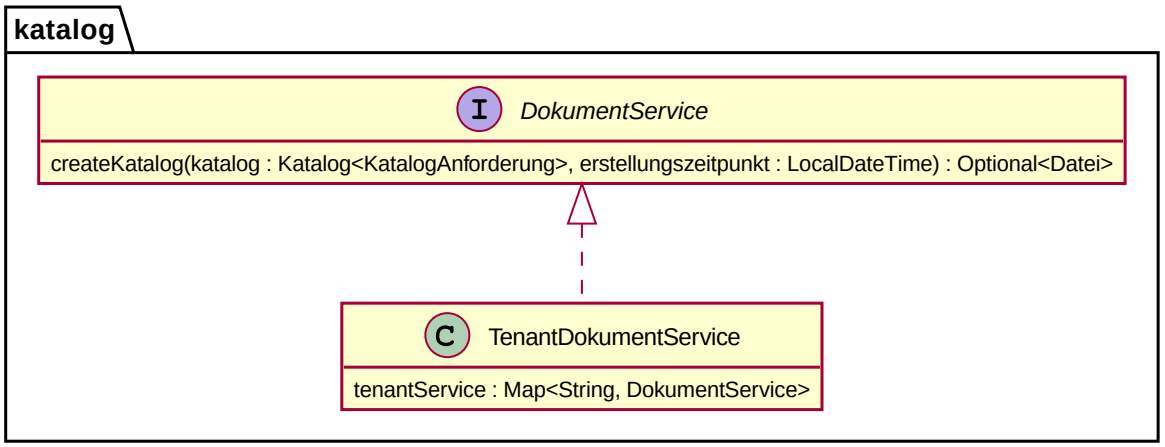

Abbildung 32. tailoringexpert-tenant:katalog

5.3.24. Whitebox *tailoringexpert-tenant:renderer*

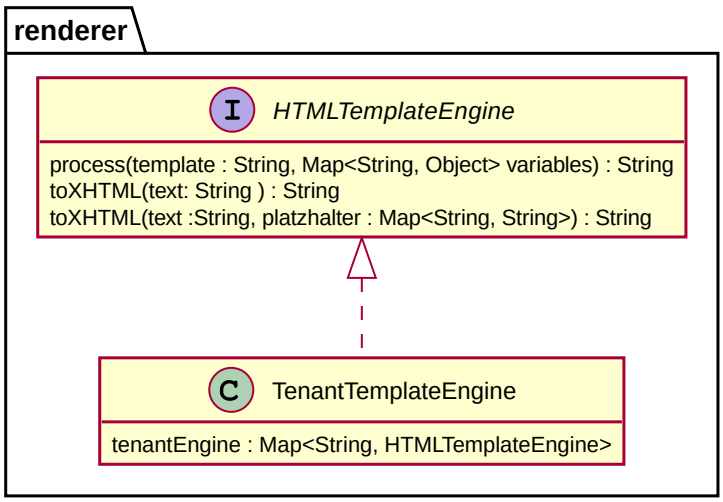

Abbildung 33. tailoringexpert-tenant:renderer

5.3.25. Whitebox *tailoringexpert-tenant:screeningsheet*

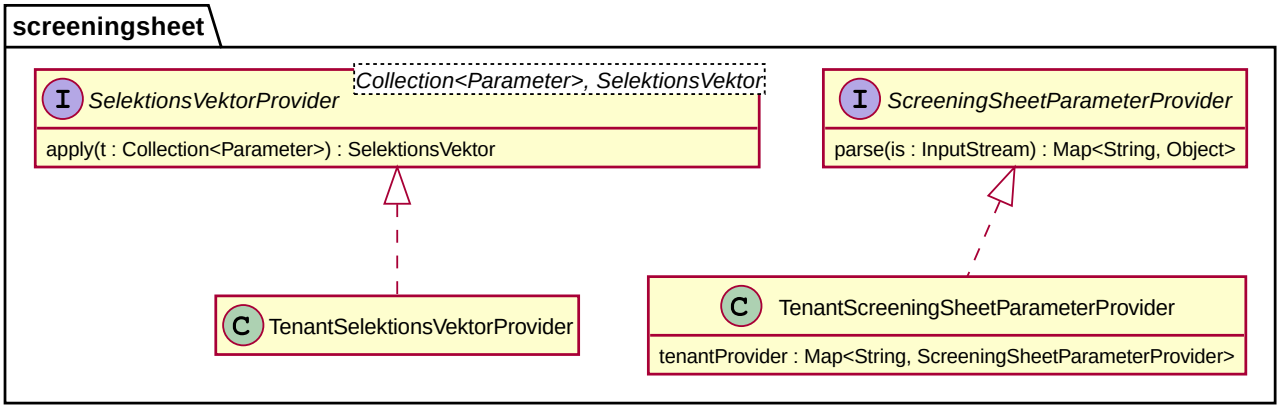

Abbildung 34. tailoringexpert-tenant:screeningsheet

5.3.26. Whitebox *tailoringexpert-tenant:tailoring*

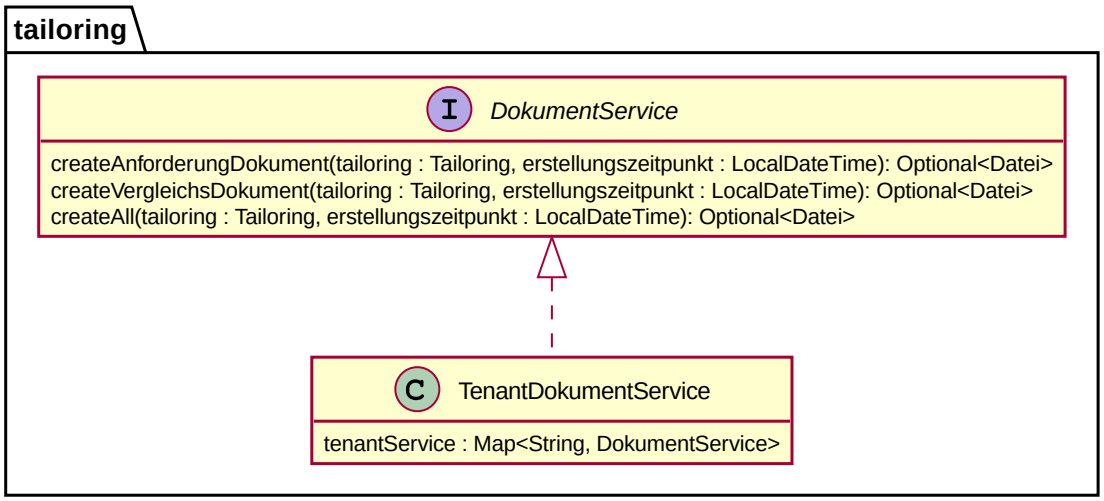

Abbildung 35. tailoringexpert-tenant:tailoring

<Whitebox-Template>

5.3.27. Whitebox <_Baustein y.1_>

<Whitebox-Template>

6. Laufzeitsicht

Die Darstellung der Sequenzdiagramme behandeln immer nur den Standardfall, dass alle Daten korrekt und in der Datenbank vorhanden sind!

6.1. Projekt anlegen

Ein Projekt wird von einem Benutzer eigentlich in zwei Schritten angelegt:

1. Hochladen und automatisiertes auswerten eines Screeningsheets
2. Eigentliches Anlegen eines Projektes mit dem anzuwendenden Selektionsvektor sowie nochmals den Rohdaten des Screeningsheets. Das Screeningsheet wird deshalb nochmals übertragen, ob manipulierte automatisch ermittelte Selektionsvektorwerte zu verhinden. Hierfür wird das Screeningsheet bei der "eigentlichen" Anlage des Projektes nochmals ausgewertet.

6.1.1. Schnittstellen Sicht

Diese Ansicht ist eine vereinfachte Ansicht. Interne Ausrufe sowie Aufrufe zu Spring Repositories und der Datenbank werden, ausser für das Speichern, in dieser Darstellung weggelassen.

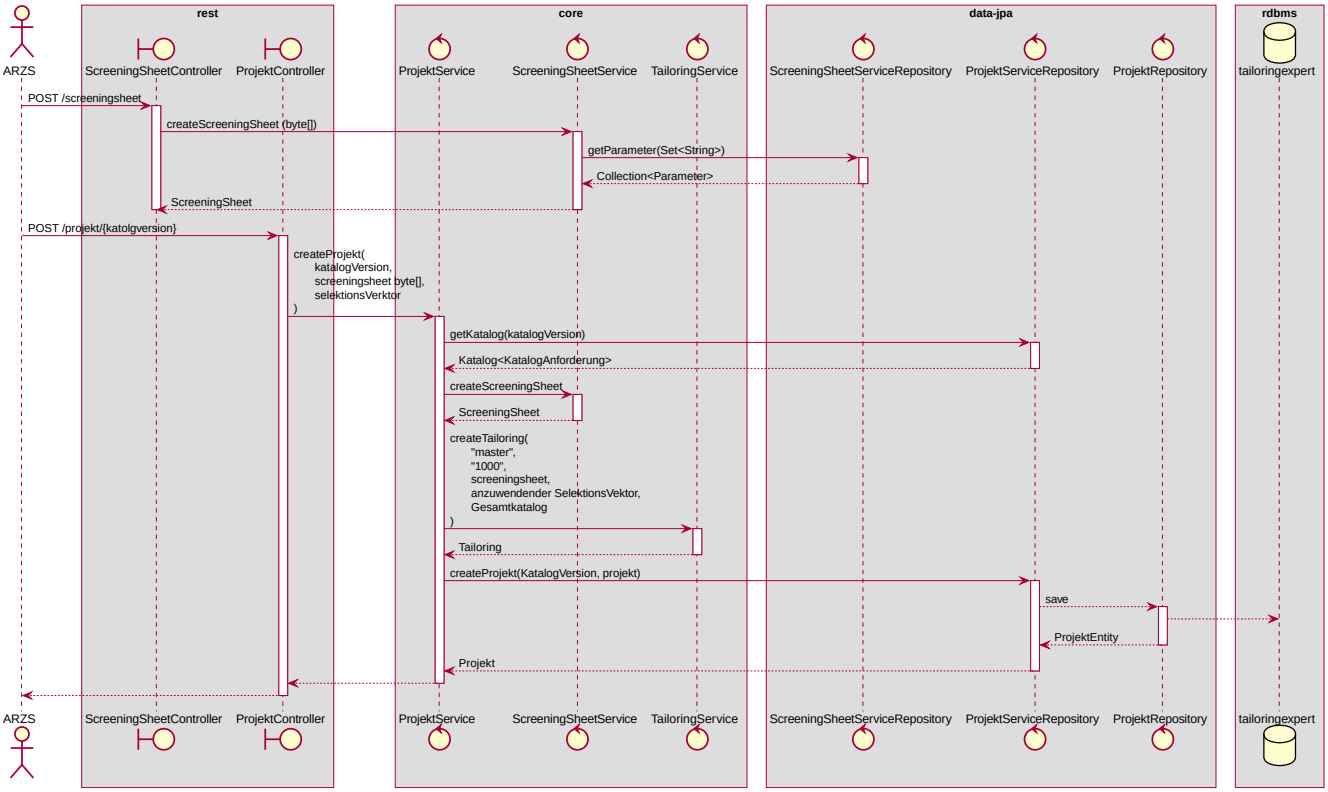

Abbildung 36. Ablauf Projekt anlegen (Schnittstellen Sicht)

6.1.2. Detailierte Sicht

In dieser Ansicht werden alle Schritte in den einzelnen Funktionen dargestellt.

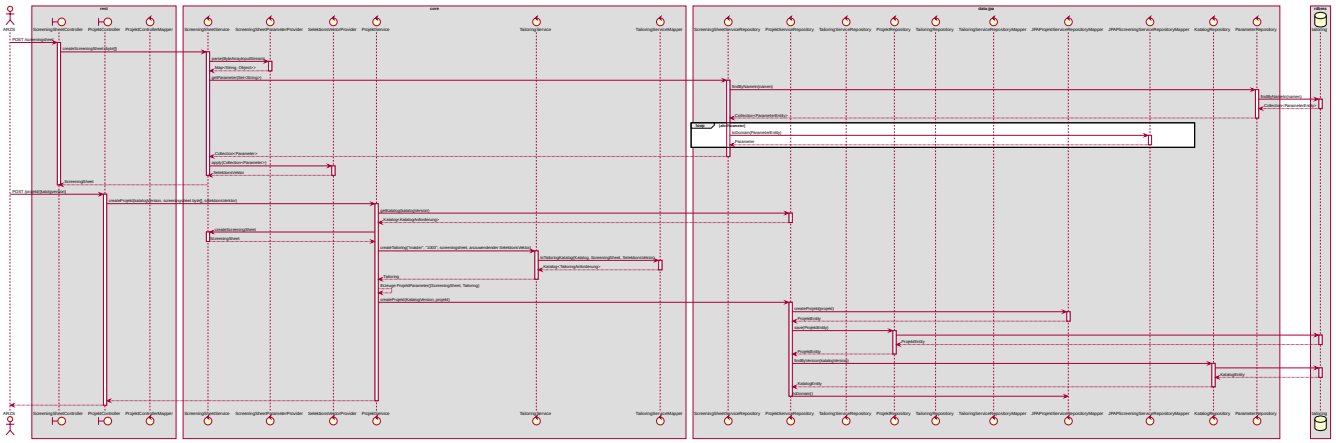

Abbildung 37. Ablauf Projekt anlegen (Detailierte Sicht)

6.2. Tailoring Anforderungen importieren

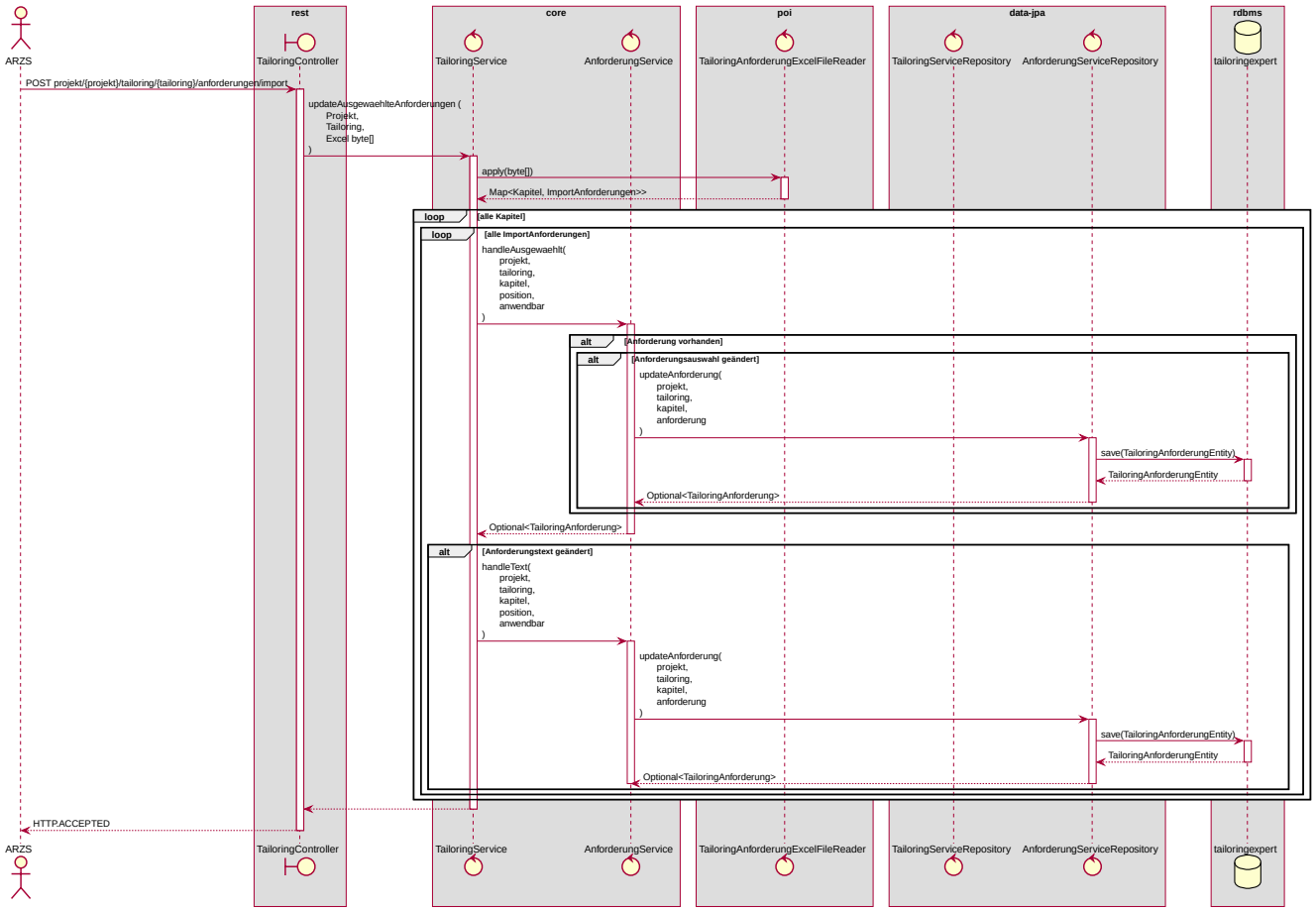

Abbildung 38. Ablauf Tailoring Anforderungen importieren

6.3. Projekt kopieren

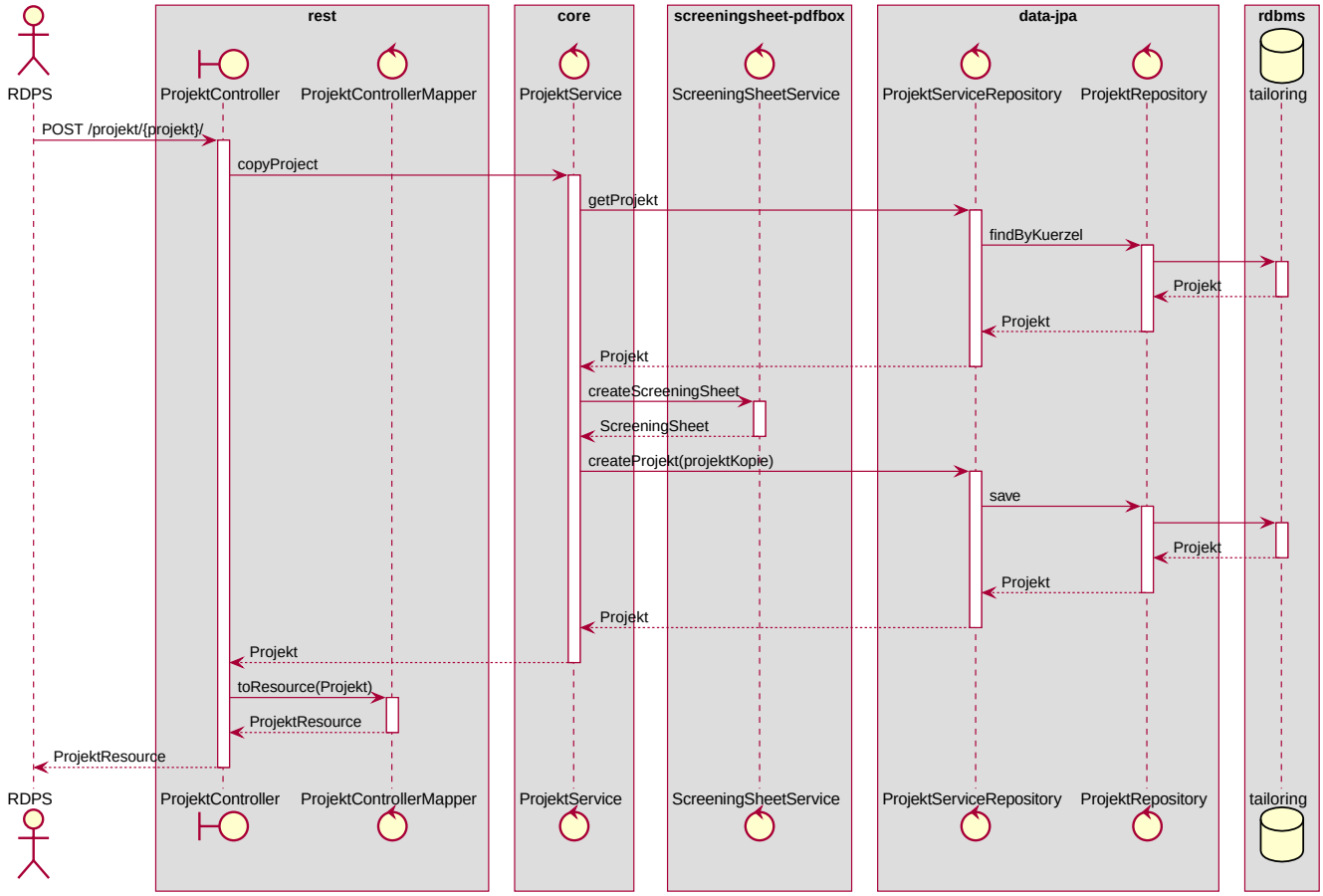

Abbildung 39. Ablauf Projekt kopieren

Bei der Kopie eines bestehenden Projektes werden alle Phasen, nicht jedoch die bereits erstellten Dokumente, mitkopiert. Für alle Phasen wird das übergebene Screeningsheet angehangen.

Der Selektionsvektor kann bei den kopierten Phasen nicht geändert werden!

6.4. Projektphase anlegen

6.5. Neue Anforderung hinzufügen

7. Verteilungssicht

7.1. Natives Deployment

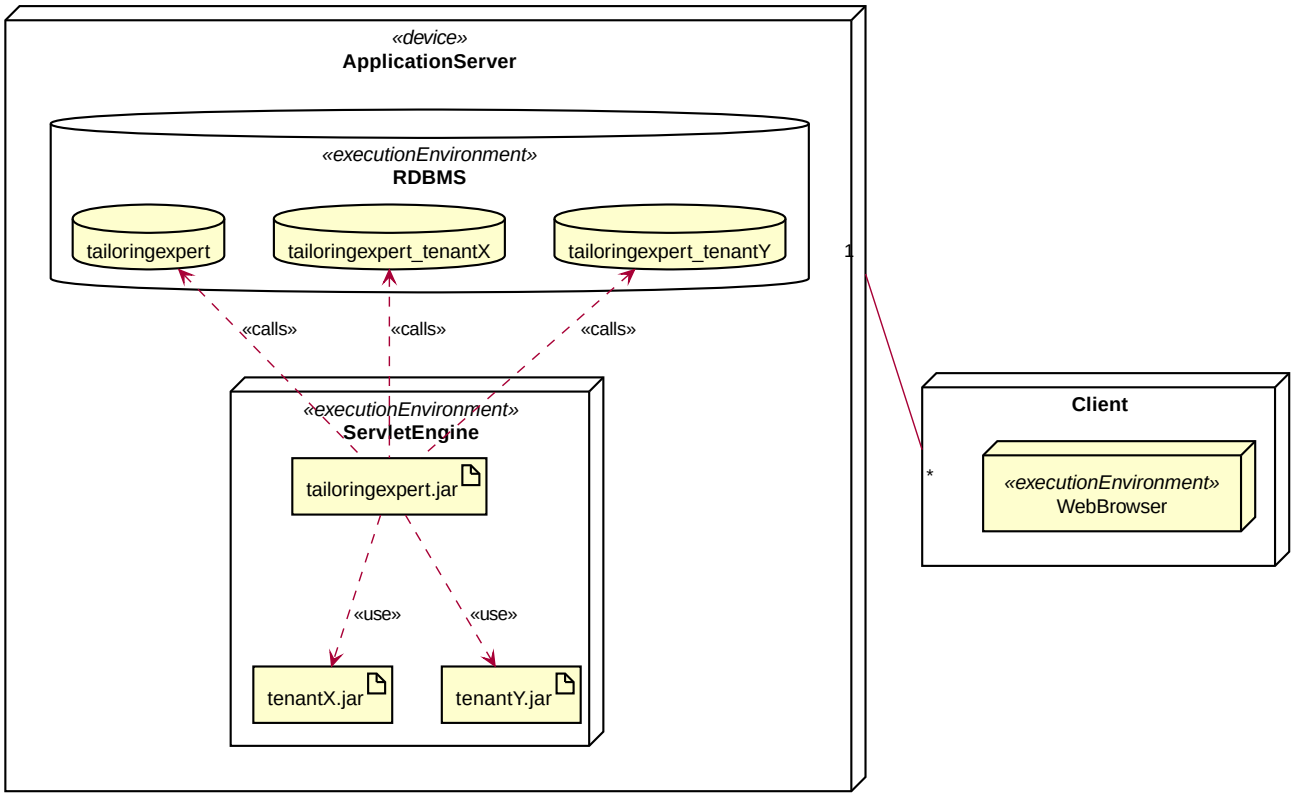

Abbildung 40. Natives Deployment

Zuordnung von Bausteinen zu Infrastruktur

Alle Module sind Teil von *tailoringexpert.jar*.

7.2. Virtualisiertes Deployment mit Docker

7.2.1. Guests

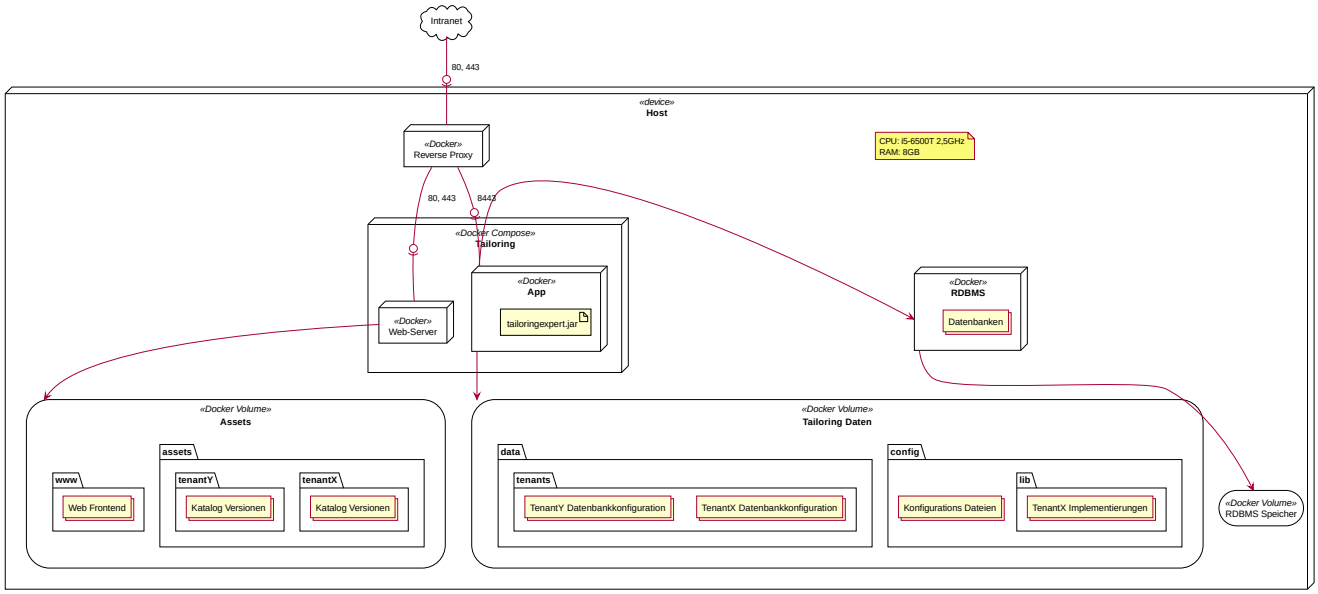

Abbildung 41. Docker Deployment

7.2.2. Netzwerke

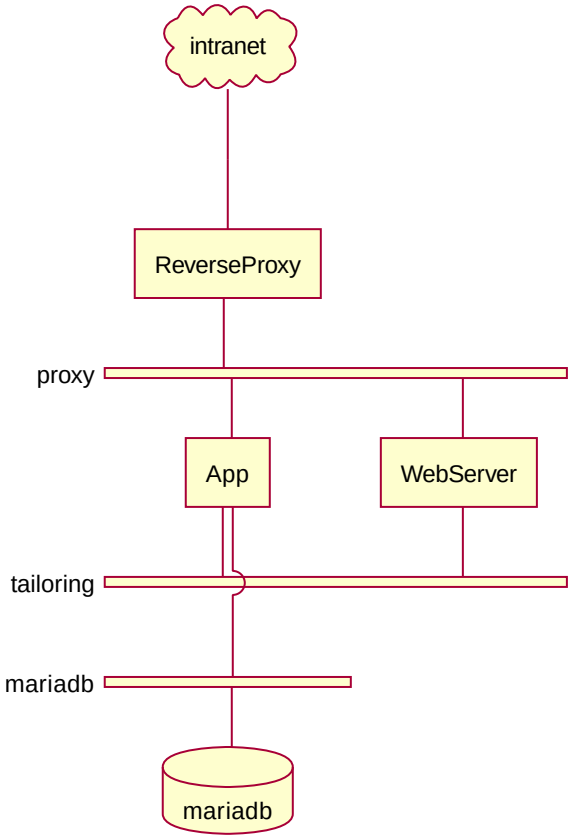

Abbildung 42. Docker Netzwerke

8. Querschnittliche Konzepte

8.1. Erzeugung von Datenobjekten

Für Datenobjekte (veränderlich/nicht veränderlich) sind Zugriffs- und Vergleichoperationen mittels Lombok zu erzeugen. Dies reduziert den zu wartenden Code erheblich. Zudem sind Methoden wie

- equals
- hashCode
- toString

konstistent und nach bewährten Mustern umgesetzt. Wenn immer es möglich ist, soll mit nicht veränderlichen Value statt mit änderbaren Datenobjekten gearbeitet werden. Für den Zugriff auf die Datenbank mit dem Persistenz-Provider ist dies nicht möglich. Für Entitäten sind deshalb immer Vakue-Objekte mit zugehörigem Builder zu erzeugen.

8.2. Datentypen

Die Domänenobjekt des fachlichen Kerns sollen keine technischen Aspekte, wie z.B. Ids für den Datenbankzugriff, enthalten. Für die Speicherung in der Persistenzschicht sollen aber technische IDs für den schnellen Zugriff auf der Datenbank enthalten sein.

8.3. Entitäten

Die Entitäten sind mittels Annotationen innerhalb des Pakets *domain* der *tailoring-data-jpa* Komponente zu erstellen. Für größtmögliche Portabilität zwischen den Datenbanken wird als *ID* -Generator eine Sequenztabelle verwendet.

8.4. Datentypkonvertierung

Konvertierungen zwischen Datentypen soll nicht manuell durchgeführt werden, da hier der Pflege- und Navigationsaufwand nicht unerheblich ist. Hierfür ist MapStruct zu verwenden. Mit MapStruct können annotationsbasiert Datenobjekte konvertiert werden.

8.5. Autorisierung und Authentifizierung

Die Anwendung implementiert keine eigene Benutzerverwaltung. Es wird davon ausgegangen, dass eine Authentifizierung mittels *.htaccess* Datei ausreichend ist.

8.6. Dependency Injection

Es soll nach Möglichkeit immer nur gegen Schnittstellen entwickelt werden. Die zu verwendenden Implementierungen der Schnittstellen werden konfiguriert. Um einen besseren Überblick zu behalten, wie die Anwendung konfiguriert ist, ist die annotationsbasierte Konfiguration grundsätzlich verboten. Für jedes Paket ist im *tailoring-bootapp*-Projekt eine Java Konfigurations

durchzuführen.

8.7. Datenbankversionierung

Für die Verwaltung der Datenbankskripte, und damit der Datenbankversion, wird liquibase verwendet. Die Datenbankskripte sind in der Komponente *tailoring-data-jpa* zu pflegen.

8.8. Testen der Architektur

Bestimmte Aspekte der Architektur werden automatisiert werden den Builds mittels archunit getestet. archunit stellt eine relativ einfache API zur Verfügung, um die wichtigsten Aspekte wie z.B.

- Paketzyklen
- erlaubte Paketzugriffe
- Verwendung von nicht erlaubten Annotationen

zu erkennen.

Zudem können bestimmte Tests mittels plantuml graphisch konfiguriert werden. Für jede Komponente ist im Basispaket ein Test *ArchitectureTest* zu definieren.

8.9. Webservices

WebServices werden als REST-Services unter Beachtung von *HATEOAS (Hypermedia as the engine of application state)* als Spring WebMVC *RestController* mittels Code-First Ansatz umgesetzt und mit **OpenApi** Annotationen dokumentiert.

8.10. Mandantenproxies

Mandantenproxies kapseln den Zugriff auf die mandantenspefischen Implementierungen Hier

```
@RequiredArgsConstructor
public class MandantenspezifischeFunktionalitaetImpl implements
MandantenspezifischeFunktionalitaet {

    @NonNull
    private final Map<String, MandantenspezifischeFunktionalitaet> tenantService;

    @Override
    public MandantenspezifischeFunktionalitaetReturnWert
mandantenspezifischeFunktionalitaet(InputStream is) {
        MandantenspezifischeFunktionalitaet provider = tenantService.get(
TenantContext.getCurrentTenant());
        return nonNull(provider) ? provider.mandantenspezifischeFunktionalitaet(is) :
null;
    }
}
```

In den Springkonfigurationen sind dies Proxies wie folgt zu definieren:

```
@Bean
@Primary
MandantenspezifischeFunktionalitaet mandantenspezifischeFunktionalitaet
(ListableBeanFactory beanFactory) {
    Map<String, MandantenspezifischeFunktionalitaet> services = Tenants.get
(beanFactory, MandantenspezifischeFunktionalitaet.class);
    return new TenantTemplateEngine(services);
}
```

```
@Tenant("tenant")
@RequiredArgsConstructor
public class TenantMandantenspezifischeFunktionalitaet implements
MandantenspezifischeFunktionalitaet {
    ...
}
```

9. Entwurfsentscheidungen

9.1. Erzeugung PA Dokumente

Status	proposed
Entscheider	Bädorf, Michael
Datum	07.03.2021
Technische Story	n/a

9.1.1. Kontext und Problemstellung

Ziel des Tailorings ist die Erzeugung von Anforderungsdokumenten. Diese Dokumente werden an den Auftragnehmer übergeben. Bisher werden diese Dokumente in Form von Word und gezeichneten PDF an die Auftragnehmer übergeben.

9.1.2. Entscheidungstreiber

- Coperate Design
- migrierte (HTML)Anforderungstextbausteine
- Hauptspeicherverwendung
- Performance

9.1.3. Betrachtete Lösungsmöglichkeiten

- Word (mittels docx4j)
- PDF (mittels openhtmltopdf)

9.1.4. Entscheidung

Gewählte Alternative

PDF (mittels openmhtmltopdf), weil

- volle HTML Fähigkeiten
- Speicherfreundlicher
- erheblich schneller

werden können

9.1.5. Vergleich der Alternativen

Word (mittels docx4j)

PA Word Dokumente können mittels Java ohne eine Word Installation erezeugt werden. Eine Library hierfür ist docx4j. Diese Library stellt eine Schnittstelle für die Erzeugung vonn Word Dateien über die XML-Schnitstelle zur Verfügung.

Postiv

- keine Word Installation nötig
- Unterstützung von HTML Inhalten
- bisherige Word Vorlage kann verwendet werden
  - Berücksichtigung Coperate Design
- erzeugtes Dokument kann nachträglich händisch bearbeitet werden
  - ggf. notwendig, wenn HTML Fehler in migrierten HTML Testbausteinen

Negativ

- XML Verrabeitung kann Speicher- und Zeitintensiv sein
- Bei der Erstellung der Dokumente werden vorhandene Grafiken/Bilddateien scheinbar nicht erinfach regerennziert sondern komplett in den Hauptspeicher geladen und in das erzeugt Word eingefügt. Diese führt spätestens bei Generierung des DRD Dokumentes zu massiven Speicherproblemen!

PDF (mittels openhtmltopdf)

PA Dokumente können mittels openhtmltopdf einfach über den "Zwischenschritt" HTML ohne CLient Installationen als PDF erezeugt werden.

Postiv

- keine Client Installation nötig
- Unterstützung von HTML Inhalten
- Formattierung/Layouting mittels CSS
- erzeugtes Dokument kann nachträglich händisch bearbeitet werden
- sehr schnell
- festes Layout/Format durch Einbettung von Schriften möglich

Negativ

- Coperate Design muss in HTML umgesetzt werden
- erzeugtes Dokument kann nicht ohne "hochwertigen" Editor manuell nachbearbeitet werden

9.1.6. Links

- docx4j

- openhtmltopdf

9.2. HTML Template Engine

Status	proposed
Entscheider	Bädorf, Michael
Datum	07.03.2021
Technische Story	n/a

9.2.1. Kontext und Problemstellung

Für die Erzeugung von PDF Dateien als HTML Code wird eine Template Engine benötigt, die Systemdaten in Vorlagen auflösen und HTML Fragmente zu einer Seite zusammenfügen kann.

9.2.2. Entscheidungstreiber

- Funktionalität
- Integrierbarkeit in das System

9.2.3. Betrachtete Lösungsmöglichkeiten

- Thymeleaf

9.2.4. Entscheidung

Gewählte Alternative

Thymeleaf, weil

- alle benötigten Funktionen zur Verfügung gestellt werden
- etabilierte sehr gute Integration in eine Spring Anwendung

9.2.5. Vergleich der Alternativen

Thymeleaf

Postiv

- einfache Template Sprache
- einfache Integration in die Anwendunf

Negativ

- n/a

9.2.6. Links

- thymeleaf

9.3. Screeningsheet

Status	proposed
Entscheider	Bädorf, Michael
Datum	07.03.2021
Technische Story	n/a

9.3.1. Kontext und Problemstellung

Die Tailoring Parameter für eine Projekt-/Phaseanlage müssen in das System eingepflegt werden.

9.3.2. Entscheidungstreiber

- Multimandantenfähigkeit

9.3.3. Betrachtete Lösungsmöglichkeiten

- Eingabe über eine Webseite
- Automatisierte Verarbeitung eines Screeningsheet

9.3.4. Entscheidung

Gewählte Alternative

Screeningsheet, weil

- Projektleiter durch seine Signatur die Korrektheit der Parameter bestätigt
- Übertragsfehler ausgegschlossen werden
- keine Oberflächen- bzw mandantenspezifischen Anpassungen erforderlich sind.
  - Implementierung erfolgt in Mandantenschnittstellen

9.3.5. Vergleich der Alternativen

Eingabe über eine Webseite

Postiv

Negativ

- Übertragsgungsfehler bei der manuellen Übernahme der Daten in das System

- Der zuständige Projektleiter muss nicht bestätigen, das die eingegeben Parameter dem Projekt und dessen Ziel entsprechen

Automatisierte Verarbeitung eines Screeningsheet

Der Systemparameter werden durch den Projektleiter in eine Datei eingetragen Diese Datei bei Projektanlage hochgeladen und automatisiert verarbeitet

Postiv

- Keine Oberflächenpflege und Weichen für Mandanten in Plattform
- Je nach vom Mandanten gewählten Dateityp kann die Korrektheit der Parameter durch Signatur des Projektleiters bestätigt werden
- Die Plattform hat keine Kenntniss über das *Wie* der einzelnen Mandanten

Negativ

9.3.6. Links

10. Qualitätsanforderungen

10.1. Qualitätsbaum

10.2. Qualitätsszenarien

11. Risiken und technische Schulden

11.1. Risiken

Beschreibung	Risko	Maßnahmen	Status

11.2. Technische Schulden

Beschreibung	Maßnahmen	Status
Die Screeningsheet Parameter sind als Datenobjekt definiert	*In Bezug auf mandantenfähigkeit sowie Erweiterbarkeit wäre es sinnvoller hier nur über eine Map zu arbeiten*	*offen*
Mandantkennung als Headerparameter	*Im Moment wird ein Default Mandant als Headerattribut in presentation-vue gesetzt Dies ist über sie initial URL zu ermitteln, im Store zu setzen und bei jedem Request zu setzen*	*offen*

12. Glossar

Begriff	Definition
Architecture Decision Records (ADR)	Architecture Decision Records (ADR) sind Templates, mit deren Hilfe grundlegende Architekturentscheidungen strukturiert festgehalten werden, so dass fundierte Entscheidungen über Änderungen auch dann getroffen unf nachvollzogen werden können.
Dependency Injection	*Als Dependency Injection (DI, englisch dependency ‚Abhängigkeit' und injection ‚Injektion', deutsch Abhängigkeitsinjektion oder Einbringen von Abhängigkeiten) wird in der objektorientierten Programmierung ein Entwurfsmuster bezeichnet, welches die Abhängigkeiten eines Objekts zur Laufzeit reglementiert: Benötigt ein Objekt beispielsweise bei seiner Initialisierung ein anderes Objekt, ist diese Abhängigkeit an einem zentralen Ort hinterlegt – es wird also nicht vom initialisierten Objekt selbst erzeugt*
docx4j	*(Open Source) Library für die Erzeugung von MS Office Dokumenten*
HTML	*HTML (HTML, englisch für Hypertext-Auszeichnungssprache) ist eine textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente wie Texte mit Hyperlinks, Bildern und anderen Inhalten.*
Java Persistence API (JPA)	*Die Java Persistence API (JPA) ist eine Schnittstelle für Java-Anwendungen, die die Zuordnung und die Übertragung von Objekten zu Datenbankeinträgen vereinfacht. Sie vereinfacht die Lösung des Problems der objektrelationalen Abbildung, das darin besteht, Laufzeit-Objekte einer Java-Anwendung über eine einzelne Sitzung hinaus zu speichern (Persistenz), wobei relationale Datenbanken eingesetzt werden können, die ursprünglich nicht für objektorientierte Datenstrukturen vorgesehen sind.*
lombok	*Project Lombok ist eine Java-Bibliothek, die hilft typischen Java Boilerplate Code für Datenobjekt annotationsbasiert statt manuell zu erstellen.*
MapStruct	*Library für die Erzeugung von Datenobjektmappings. Die Mappings werden zur Compile-Zeit aus im Code befindlichen Annotation zu Java Quellcode konvertiert. Somit ist das erzeugte Mapping auch debugbar*

Begriff	Definition
Spring Data JPA	*Spring Data JPA erleichtert die einfache Implementierung von JPA-basierten Repository und smiut das Erstellen von Spring-basierten Anwendungen, die Datenzugriffstechnologien verwenden.*
openhtmltopdf	*Library für die Erzeugung von PDF Dokumenten aus HTML Code.*