Technische Dokumentation Sanktionslisten

Projektteam Sanktionslisten

Dario Carosella
Samuel Ackermann

V 1.0 / 22.12.2017

Inhaltsverzeichnis

Zweck des Dokuments

Dieses Dokument beschreibt kurz den Ablauf sowie Umsetzung des Projektes Sanktionslisten. Am Ende des Dokumentes ziehen wir ein Schluss Fazit und machen einen Soll-Ist-Vergleich.

Projektidee

Das Projekt Sanktionslisten soll ein Produkt hervorbringen, welches verschiedenen Firmen und Privatpersonen erlaubt herauszufinden, ob sie mit einer Person/Organisation eine Geschäftsbeziehung eingehen dürfen oder nicht. Das Produkt soll einen Webservice beinhalten, welcher einfach in bestehende Programme/Services integriert werden kann. Weiters soll ein einfacher Client bereitgestellt werden, welcher direkt verwendet werden kann und vor allem für Privatpersonen gedacht ist.
Heutzutage ist die Suche von sanktionierten Organisationen oder Personen relativ aufwändig, da die genannten Listen weitläufig verteilt sind und nicht zentral verwaltet werden. Und genau bei diesem Punkt soll der Hauptnutzen sein. Mit dem Import von verschiedenen Quellen sollen diese Listen zentralisiert und dem Endbenutzer in einer praktischen Sicht dargestellt werden. Zudem wird angestrebt, dass der Validierungsprozess von sanktionierten Personen vereinfacht oder sogar automatisiert werden kann.

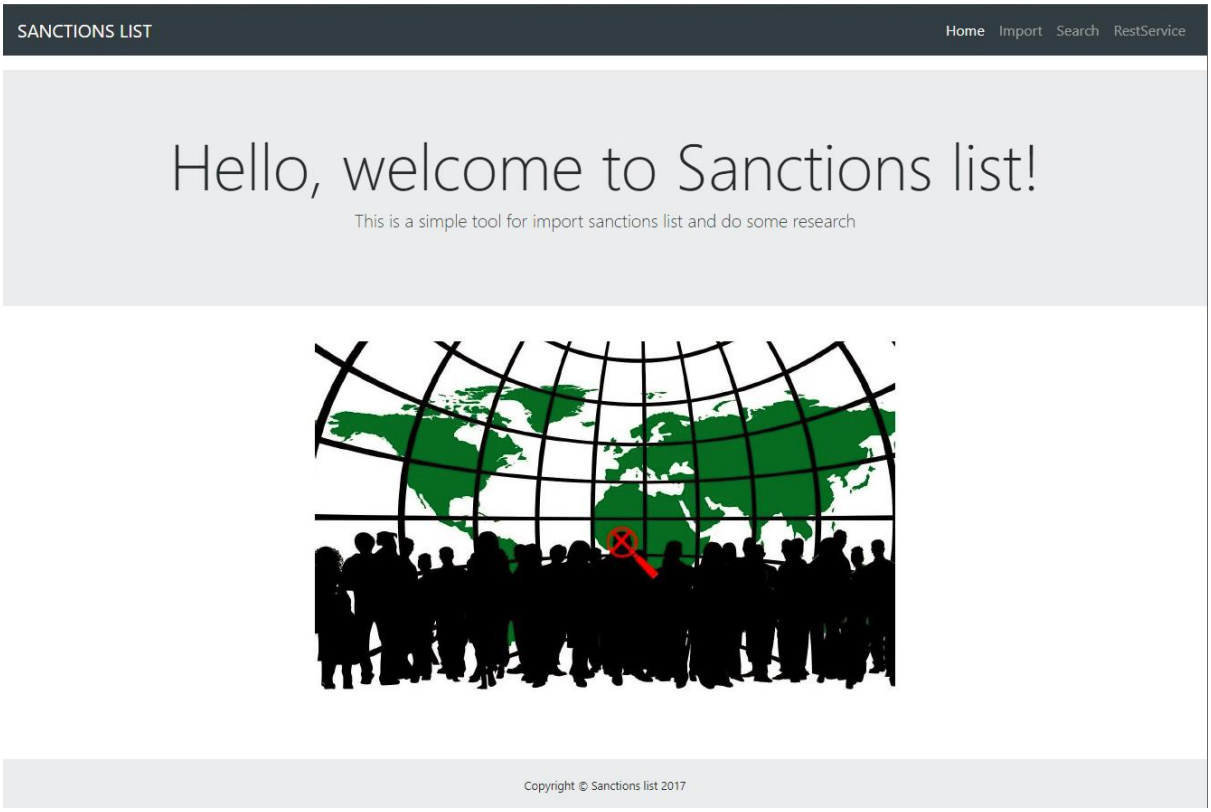


Figure 1: Homepage

Vorgehensweise

Organisatorischer Ablauf

Das Projekt realisierten wir in drei Phasen. Die erste Phase war die Projektfindung respektive das Schreiben der Anforderungsspezifikation. In einer zweiten Phase folgte die Umsetzung. Schliesslich kamen wir am Ende zur dritten und letzte Phase. In dieser Phase stand das schreiben dieses Dokumentes, so wie Analyse der getanen Arbeit im Vordergrund.

Erste Phase

Bevor wir mit der Realisierung des Projektes starteten, befassten wir uns mit der Anforderungsspezifikation. Für diese führten wir mehrere Gespräche mit Herrn Kaltz um uns einen Überblick zu erarbeiten, was von uns gefordert wird. Auch das Internet diente uns als Recherche um die Anforderungen auf Papier zu bringen. Auch ein wichtiger Punkt in dieser Phase war das bestimmen der Technologien. Wir einigten uns, dass wir das Frontend in Anugular/Typescript und das Backend mittels Java realisieren.

Zweite Phase

Nachdem die Anforderungsspezifikation von Herrn Kaltz und Herrn Schwab abgenommen worden war, starteten wir sogleich mit der Realisierung des Projektes. Wir starteten unsere Programming-Session jeweils mit einem kurzen Abgleich/Stand-up-Meeting, sowie einen Ausblick was wir heute erreichen wollen. Auch entstand in dieser Phase die Architektur unserer Software. Das Datenbankschema erfolgte aus der Datenanalyse der beiden Sanktionslisten von der EU sowie der SECO. Die gesamte Architektur, sowie Implementation wurde so umgesetzt, dass es jederzeit möglich ist den Import und somit auch die Suche der Sanktionen durch weitere Quellen zu ergänze. Mehr dazu können Sie im Kapitel Umsetzung entnehmen.

Für das Backend und somit unter anderem auch für das Zusammenführen der zwei Sanktionslisten (EU & SECO) und für die Suche, war Samuel Ackermann zuständig. Für das Frontend, sowie zur Verfügung stellen der Webservices, war Dario Carosella zuständig. Durch diese Aufteilung der Aufgaben war es uns möglich autonom zu arbeiten. In den letzten Tagen der Phase zwei, stand dann das Zusammenführen der beiden Implementationen im Vordergrund.

Dritte Phase

In der dritten Phase stand das Analysieren der getanen Arbeit im Vordergrund. Auch entstand in dieser Phase dieses Dokument, wo wir nun kurz aufzeigen möchten, wie wir unsere Arbeit aufgeteilt haben und was wir alles erreichen konnten und was es noch zu verbessern gibt. Auch möchten wir in diesem Dokument kurz erläutern was es für Erweiterungsmöglichkeiten gibt. Zudem gehört auch die Vorbereitung, sowie das Präsentieren des Projektes zur dritten Phase.

Technische Unterstützung

Git - Versionsverwaltung

Wir haben uns als Versionsverwaltungssystem für Git entschieden, da diese Software kinderleicht zu bedienen ist. Zudem kann man von jedem beliebigen Rechner auf den Sourcecode zugreifen. So war es mir, Dario Carosella, möglich von meinem Arbeitsnotebook (der deutlich mehr Power hat als mein Privates), sowie mit meinem privaten Notebook zu entwickeln. Ich musste einfach nur einen Branch erstellen und jeweils meine Änderungen committen und pushen damit ich diese danach auf meinem anderen Notebook weiterfahren konnte. Schliesslich wenn ein Feature fertig entwickelt wurde, pushte man die “finale Version” auf seinem Remote Branch und eröffnete auf GitHub einen Pull request.

Jenkins - Kontinuierliche Integration

Sobald ein PullRequest eröffnet wird, startet auf Jenkins automatisch ein Job welcher der eingecheckte Code auf Herz und Nieren testet. Konkret heisst das bei uns, dass der Code auf die Softwarequalität, Lesbarkeit und Wiederverwendbarkeit überprüft wird. All dies haben wir in einem Checkstyle definiert. Falls mehr als 20 kleinere Checkstyle warning auftreten failed der Job und der PullRequest ist somit nicht merge bar. Damit stellen wir fest, dass die Qualität unseres Codes auf ein bestimmtes Niveau ist. Auch die Unit-Test werden überprüft ob alle “grün” sind. Zu Guter letzt wird noch überprüft ob der Code kompilierbar ist. Ist alles erfüllt, so ist der Job grün, also OK und der Pull request darf gemerged werden.

GitHub - Filehosting

Unser Sourcecode haben wir auf GitHub abgelegt. Wir haben für unser Projekt zwei Repositories erstellt. Eines für das Frontend und ein zweites für das Backend. Auf GitHub hat man einen Überblick über alle Branches und somit über den gesamten Code. Hauptsächlich nutzen wir aber GitHub für die Pull requeste zu eröffnen. Hat einer von unse ein Feature fertig implementiert, so eröffnet er einen sogenannten PR und fügt den anderen als Reviewer hinzu. Nun wird wie oben beschrieben der Code zuerst von Jenkins analysiert. Anschliessend hat der Reviewer die Möglichkeit die Änderungen einzusehen und zu kommentieren, wenn er etwas anders gemacht hätte oder mit dem Ergebnis nicht zufrieden ist. Ist alles in Ordnung so wird der PR gemerged. Allerdings darf nur der Reviewer mergen, der Owner hat keine Rechte seinen eigenen PR zu mergen.

Umsetzung

Frameworks

Wir haben folgende Frameworks bei unserem Projekt eingesetzt.

Wildfly Swarm

WildFly Swarm ist ein sogenannter Microservices. Dieser verpackt genau so viel vom Applikationsserver mit der eigentlichen Anwendung zusammen wie nötig ist, um ein ausführbares Java-Archiv zu erhalten. Auch “single jar application “ oder “fat jar”-Konzept genannt.

RESTEasy

RESTEasy ist ein JBoss-Projekt, das verschiedene Frameworks für die Erstellung von RESTful Web Services und RESTful Java-Anwendungen bereitstellt. Es ist eine vollständige zertifizierte sowie portable Implementierung der JAX-RS 2.0 Spezifikation, die eine JAVA API für RESTful Web Services über das HTTP-Protokoll bereitstellt.

Multipart

Multipart ist Package vom RESTEasy Framework. Dieses wird gebraucht um Files auf unseren Backend zu speichern. Der POST Aufruf von der Website übergibt alle nötige Daten aus dem Webformular dem Webservice. So kann danach mithilfe von Multipart der Filename herausgelesen werden und schliesslich korrekt abgespeichert werden.

H2DB

Die H2 Datenbank ist ein relationales Datenbankmanagementsystem das mit Java entwickelt wurde. Das System wird als JAR direkt in die Anwendung eingebettet. Als Schnittstelle zur Ansprache der mit dem System betriebenen Datenbanken steht SQL und JDBC zur Verfügung

Crnk

Crnk, ausgesprochen Crank, ist eine Implementierung der JSON-API-Spezifikation um das Erstellen von RESTful-Anwendungen zu erleichtern.

Batch JBeret

Java Batch ist ein Standard, welches erlaubt grosse Daten zu Verarbeiten. Die konkrete Implementierung dieses Standards übernimmt die Library JBeret. Die ganze Job Umgebung ist in JBeret implementiert. In unserem Projekt müssen wir lediglich den Job definieren und die dazugehörigen Schritten.

Samsamann

Ist eine eigene von Samuel Ackermann geschriebene Library. Sie erleichtert, bereits existierende Entities mit Hilfe von Crnk als Rest-Endpunkte anzubieten. Alle unsere Entities erben der BaseEntity Klasse. Diese Klasse stellt eine eindeutige ID, sowie ein “creationDate” und “updateDate” zur verfügung.

Typescript

Im Frontend brauchen wir nebst dem normalen HTML5 und Anular noch das Typescript Framework. TypeScript wurde von Microsoft entwickelt. Mit Typescript ist unter anderem eine Typisierung möglich. Auch lassen sich Klassen sowie Vererbungen definieren.

Docker

Docker dient dazu, Anwendungen in Container zu isolieren und damit sicherer, portabler und vor allem austauschbar zu gestalten. Dabei basiert Docker auf Linux-Techniken, wird jedoch unter anderem auch von Windows 10 und Windows 2016 Server unterstützt. Unser WildFly Swarm JAR läuft in der Produktiven Umgebung auf Docker.
Architektur

Systemarchitektur

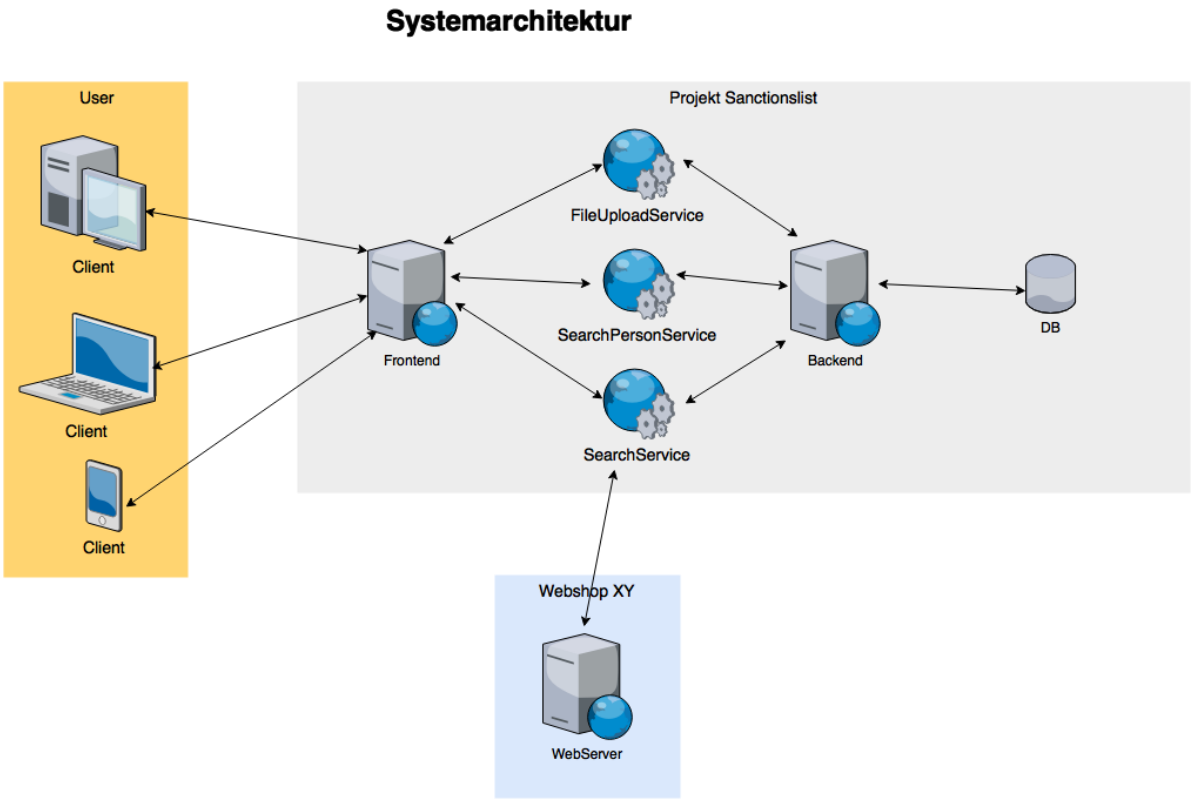


Figure 2: Systemarchitektur

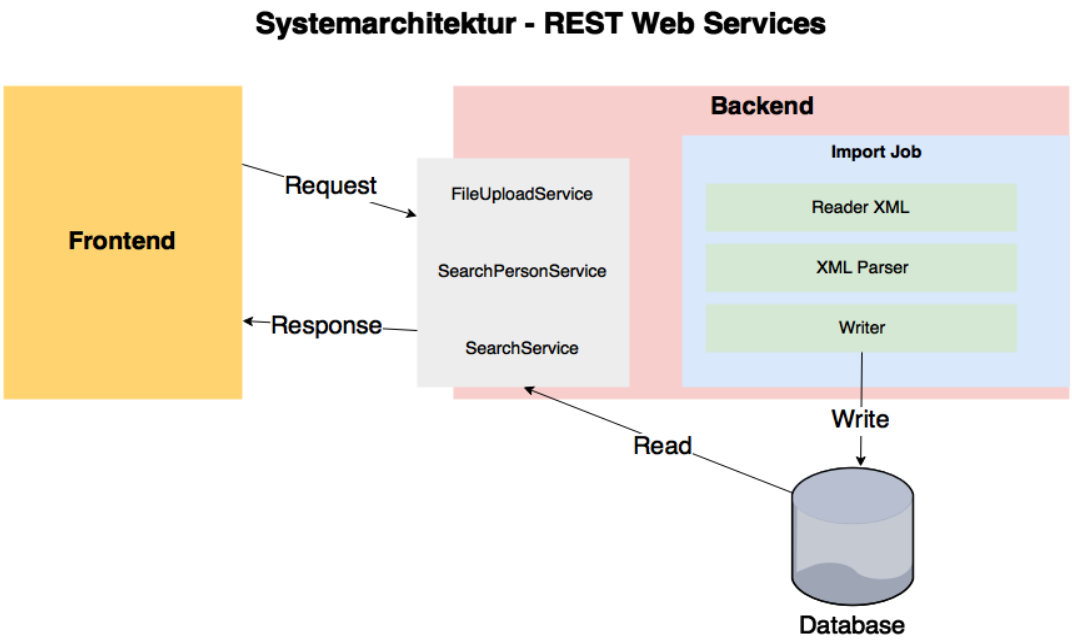
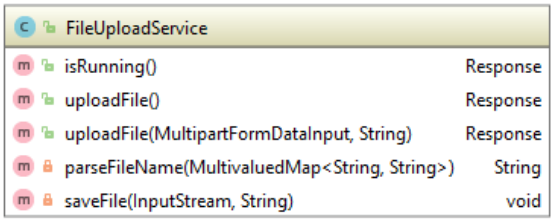
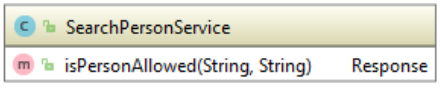


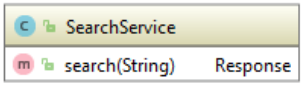
Figure 3: Systemarchitektur "REST Web Service"

Klassendiagramm

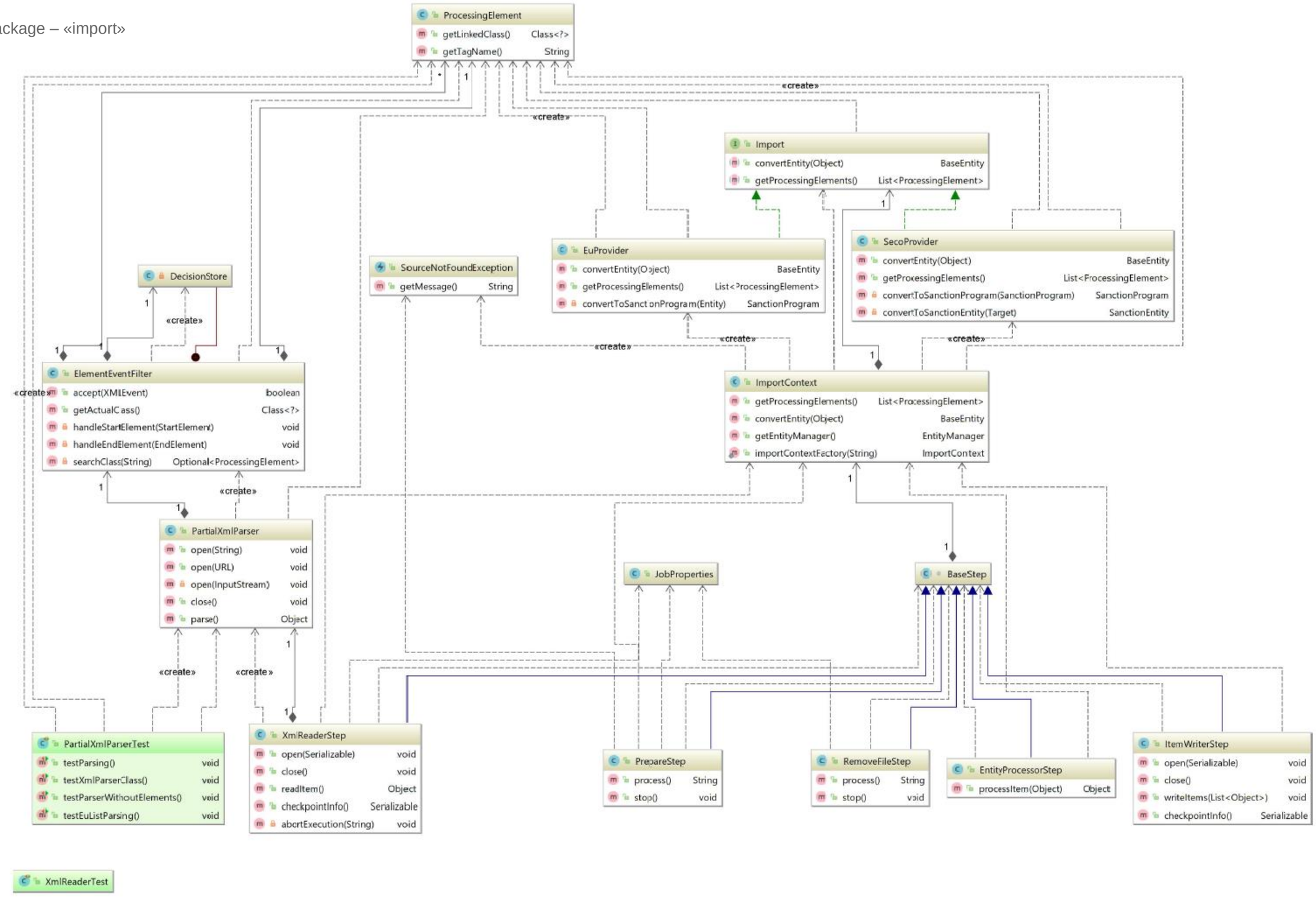
Package – «rest»



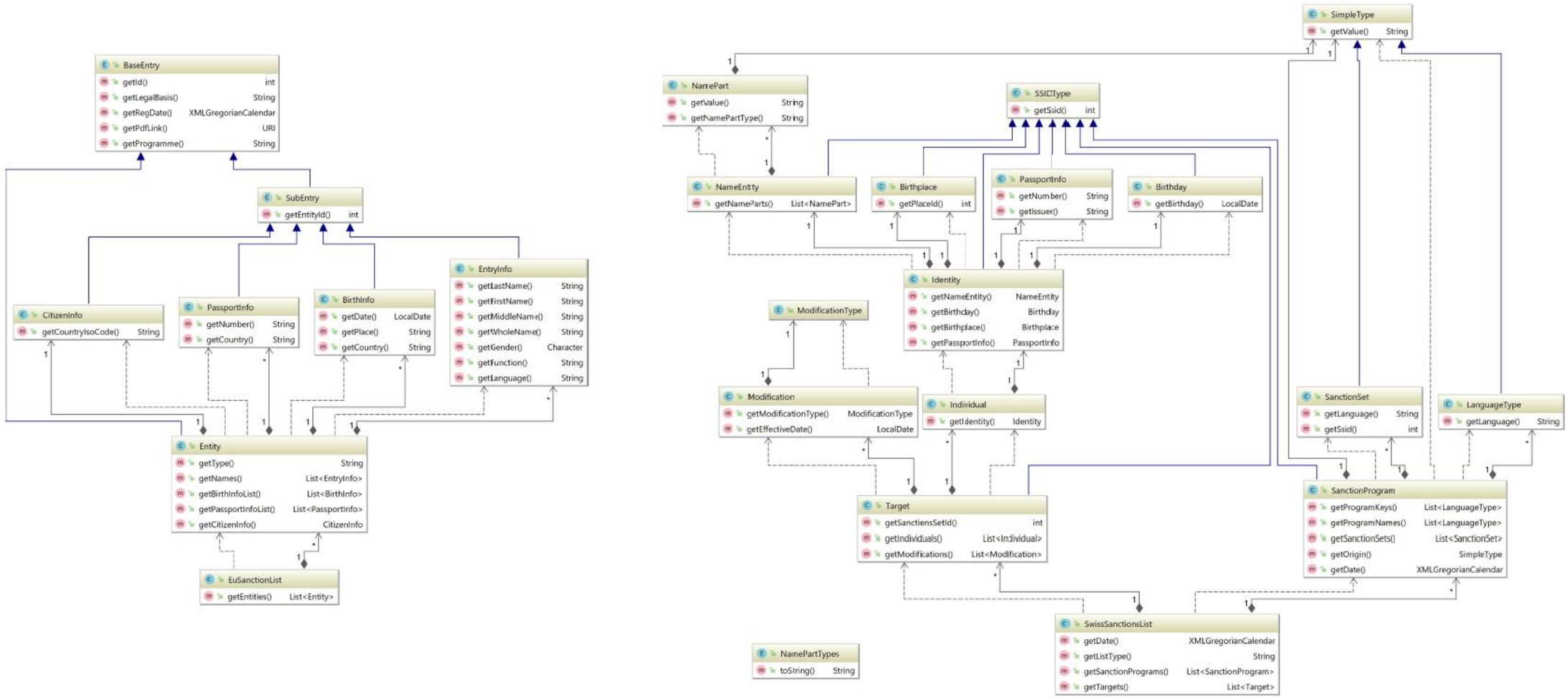




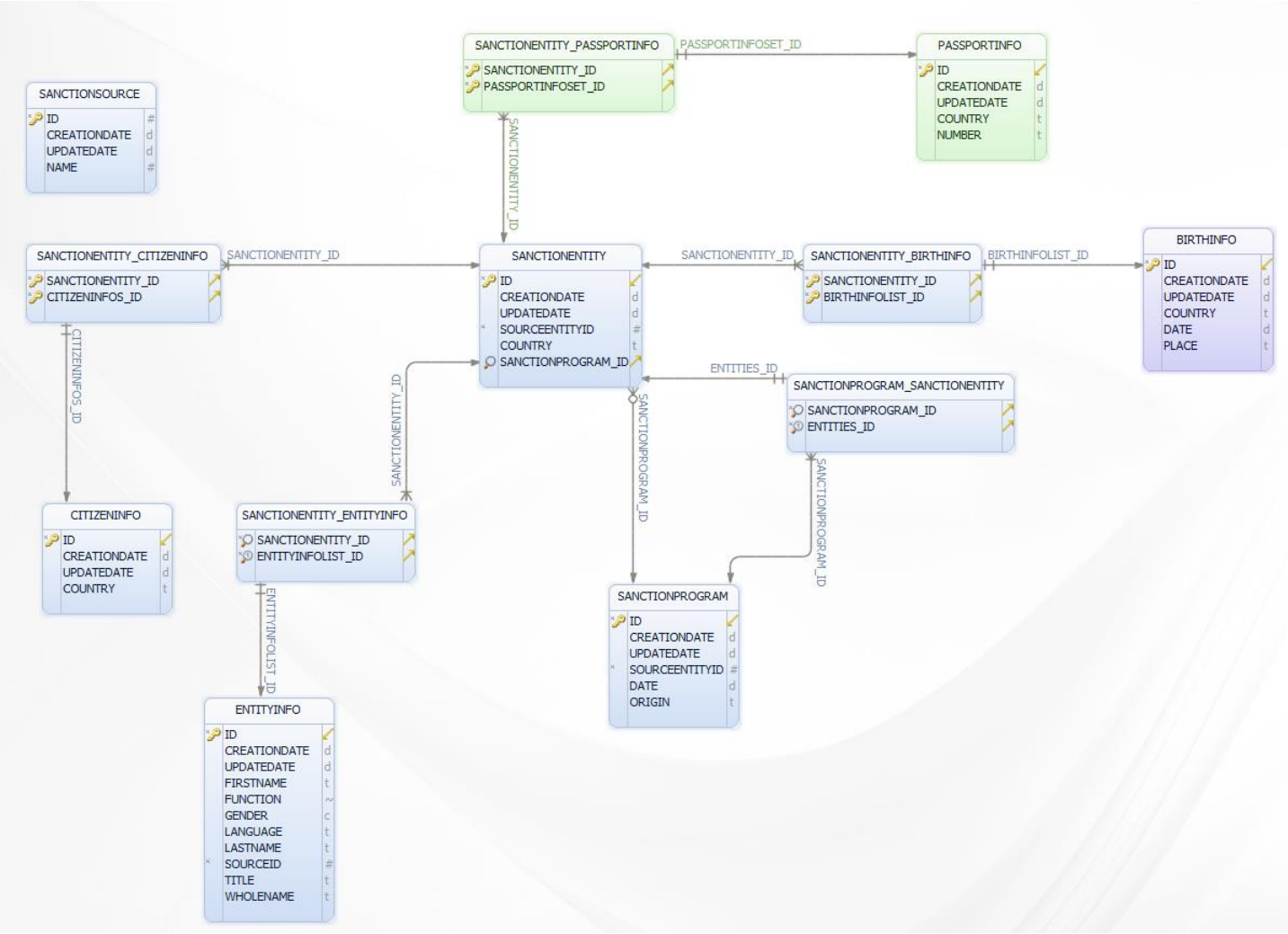
Package – «import»

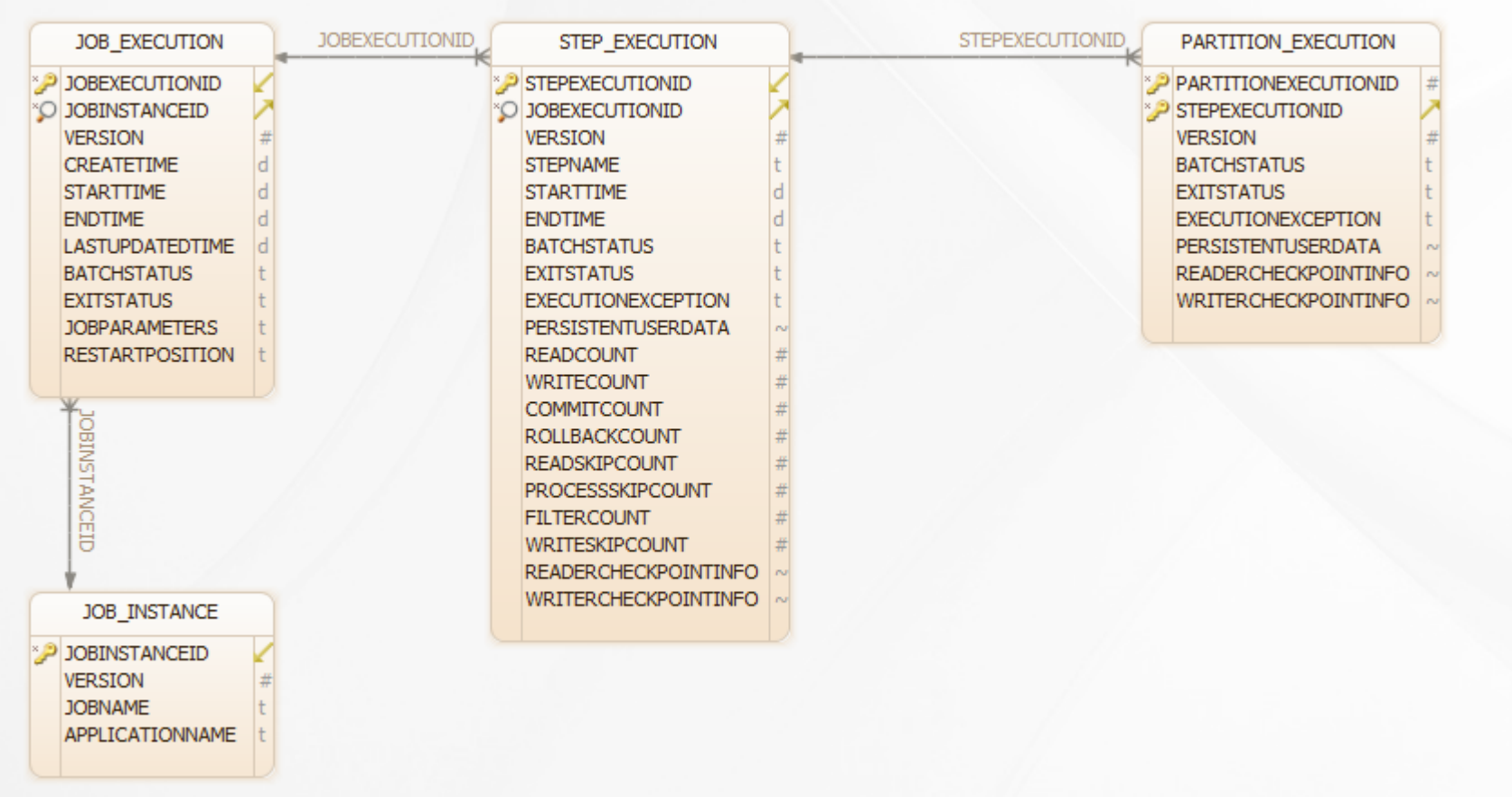


Package – «provider»



Datenbankdiagramm




JOB_EXECUTION
JOBEXECUTIONID
JOBINSTANCEID
VERSION
CREATETIME
STARTTIME
ENDTIME
LASTUPDATEDTIME
BATCHSTATUS
EXITSTATUS
JOBPARAMETERS
RESTARTPOSITION
JOBEXECUTIONID
STEP_EXECUTION
STEPEXECUTIONID
JOBEXECUTIONID
VERSION
STEPNAME
STARTTIME
ENDTIME
BATCHSTATUS
EXITSTATUS
EXECUTIONEXCEPTION
PERSISTENTUSERDATA
READCOUNT
WRITECOUNT
COMMITCOUNT
ROLLBACKCOUNT
READSKIPCOUNT
PROCESSSKIPCOUNT
FILTERCOUNT
WRITESKIPCOUNT
READERCHECKPOINTINFO
WRITERCHECKPOINTINFO
STEPEXECUTIONID
PARTITION_EXECUTION
PARTITIONEXECUTIONID
STEPEXECUTIONID
VERSION
BATCHSTATUS
EXITSTATUS
EXECUTIONEXCEPTION
PERSISTENTUSERDATA
READERCHECKPOINTINFO
WRITERCHECKPOINTINFO
JOBINSTANCEID
JOB_INSTANCE
JOBINSTANCEID
VERSION
JOBNAME
APPLICATIONNAME

Schlussbericht

Soll-Ist-Vergleich

Der Soll-Ist-Vergleich wird mit den Funktionalen, Technischen sowie Qualitätsanforderungen aus der Anforderungsspezifikation verglichen. In der ersten Spalte ist jeweils die ID, in der zweite die Priorität der Anforderung. Die Beschreibung ist das Soll und die Umsetzung ist mit dem Ist gleichzusetzen.

Funktionale Anforderungen

Table 1: Funktionale Anforderung 1

ID	PRIO	BESCHREIBUNG	UMSETZUNG
F1	M	Einlesen von verschiedenen Sanktionslisten (XML-Quelldateien)	Über die Importfunktion können verschiedene Sanktionsliste importiert werden.
F1.1	M	Die Sanktionslisten (.xml files) sollen geparst werden und in eine Datenbank abgeglichen.	Die Sanktionslisten (.xml files) werden in einer h2db abgespeichert.
F1.2	M	Es sollen von Beginn mindestens 2 oder mehr Quellen unterstützt werden und importiert werden können.	Aktuell ist es möglich die Sanktionslisten von der EU sowie SECO einzulesen. Weitere Quellen sind mit kleineren Codeanpassungen möglich.
F1.3	M	Die importierten Daten sollen zusammengefasst werden und als einheitliche Quelle fungieren für den Webservice	Die beiden Quellen werden zusammengeführt und in der DB abgespeichert.
F1.4	M	Das Backend soll Änderungen erkennen können und die entsprechende Einträge updaten.	Wir haben probiert mit Hilfe von JPA/EntityManager dies zu realisieren. Scheiterten jedoch an einer eindeutigen Identifikation.
F1.5	O P1	Die eingelesenen Dateien sollen gegenüber einem Schema validiert werden.	Aktuell werden die Dateien nicht validiert.

Table 2: Funktionale Anforderung 2

ID	PRIO	BESCHREIBUNG	UMSETZUNG
F2	M	Die importierten Daten sollen via Webservice zur Verfügung stehen	Die Daten können via Webservice abgefragt werden.
F2.1	M	Der Webservice definiert einen Endpunkt bei dem via GET-Request eine Suchanfrage gestartet werden	Dem Webservice können zwei Parameter (Vor- und Nachname) übergeben werden. Nach diesen Parametern wird anschließend gesucht.
F2.2	M	Das Suchresultat soll verschiedene Metadaten liefern, die zu einem besseren Resultat führen.	Die Suche nach Vor- und Nachname liefert, fall es für die Person einen Eintrag gibt, das gesamte Entity zurück.

Table 3: Funktionale Anforderung 3

ID	PRIO	BESCHREIBUNG	UMSETZUNG
F3	M	Verwalten der Daten über ein Admintool.	Das Admintool, eine Webapplikation, beinhaltet die Importfunktion sowie eine Suchfunktion.
F3.1	O P1	Mittels Volltextsuche kann die Datenbank nach Sanktionen durchsucht werden.	Die Datenbank kann mittels einer Volltextsuche durchforscht werden.
F3.2	M	Die Quellen von Sanktionslisten können über das Admintool verwaltet (Aktualisieren, Neu einlesen) werden.	Es gibt im Admintool eine Seite wo man ganz einfach eine neue Sanktionsliste importieren kann.
F3.3	W	Zusätzlich zu den sanktionierten Personen/Institutionen werden auch noch die zugehörigen Gesetzestexte und/oder Sanktionsmassnahmen angezeigt.	Es werden nur die sanktionierten Personen/Institutionen angezeigt.

Technische Anforderungen

Table 4: Technische Anforderung

ID	PRIO	BESCHREIBUNG	UMSETZUNG
T1	M	Der Webservice soll mittels Access-Token geschützt werden, damit nur autorisierte Personen darauf zugreifen können.	Leider konnte diese Anforderung nicht umgesetzt werden. => Zeitmangel
T2	M	Die importierten Daten sollen in einer Datenbank gespeichert werden.	Wurde umgesetzt.
T3	M	Der Import der Daten soll losgelöst des Webservices in einem Job realisiert sein.	Der ImportService kickt schliesslich den Import Job an. Der Job kümmert sich um das Zusammenführen sowie importieren der Daten in die Datenbank.
T4	O P1	Das Admintool soll mittels Login geschützt werden.	Wurde nicht umgesetzt.
T5	O P1	Mittels Caching sollen die Suchanfragen schneller beantwortet werden können	Die Suchanfragen werden nicht gecacht.
T6	W	Das Backend soll mittels Docker ausgeliefert werden.	Das Backend wird mittels Docker ausgeliefert.

Qualitätsanforderungen

Table 5: Qualitätsanforderungen

ID	PRIO	BESCHREIBUNG	UMSETZUNG
Q1	M	Der Quellcode soll mittels Unit-Tests getestet werden. Es soll eine minimale Abdeckung von 80% erreicht werden.	Wurde umgesetzt.
Q2	M	Der geschriebene Programmcode soll in GIT(SCM) verwaltet werden.	Wurde umgesetzt.
Q3	W	Der Sourcecode soll mittels Continuous Integration Tool fortlaufend auf Qualitätsmerkmale getestet werden.	Mittels verschiedene Checks wird der Code im Jenkins auf Herz und Nieren getestet.

Attributlegende

- ID: eindeutige Identifikation
- PRIO: **M**uss / **O**ptional P1, P2, P3 / **W**unsch (Nice to have)
- BESCHREIBUNG: SOLL
- UMSETZUNG: IST
- ROT: Nicht erfüllt/umgesetzt

Team Fazit

Die letzten Projekttage waren lange und Anstrengend… Warum, war da etwa eine Fehlplanung vorhanden? Hatten wir das ganze Zeitlich ein bisschen unterschätzt? Haben wir uns zu viel vorgenommen?

Ich glaube wir können fast alle Fragen mit einem JA beantworten. Erstens waren wir mit der Planung etwas zu optimistisch. Zu einem hatten wir grössere Mühe, als vorgängig angenommen, mit der Import Geschichte. Die beiden .xml Files von der SECO und EU sind riesig und vor allem komplett unterschiedlich. Dies brauchte viele Denkarbeit und Anläufe um diese sauber Zusammenführen zu können. Doch schliesslich hat es doch zufrieden stellen geklappt. Hingegen brauchten wir, Hibernate sei Dank, für die Suche viel weniger lange. Da Hibernate Search bereits eine Full-Text Suche für Entitäten anbietet war dies viel einfacher umsetzbar als während der Projektplanung angenommen. Das zweite was uns Mühe bereitete war die Unerfahrenheit im Bereich Frontend. Konkret die fehlenden Kenntnisse mit Angular/Typescript. Wir wollten das Projekt 1 nützen um eine neue Interessante Programmier-/Scriptsprache kennen zu lernen. Das einarbeiten hatte doch um einiges mehr Zeit in Anspruch genommen als wir gedacht haben.

Ich glaube nicht, dass wir das Ganze zeitlich unterschätzt haben. Viel mehr haben wir die Zeiten falsch eingeplant. So wäre es besser gewesen, wenn wir schon zu Beginn des Projektes uns an den Wochenenden zum Programmieren in der Schule getroffen hätten und nicht erst gegen Ende der Projektarbeit.

Ich glaube nicht, dass wir uns zu viel zugemutet haben, wie schon erwähnt war das grösste Problem das Zeitmanagement. Das müssen wir in den nächsten Projekten die wir angehen besser machen.

Im Grossen und Ganzen denke ich aber, können wir mit dem Endresultat zufrieden sein. Wir konnten fast alle Features so umsetzten, wie wir sie in der Anforderungsspezifikation beschrieben haben (Siehe Kapitel Soll-Ist-Vergleich). Auch konnten wir viele neue Frameworks entdecken, welche uns in unserem Berufsalltag von Nutzen sind. Wir konnten in Kurs Projekt 1 ein eigenes Produkt abliefern und sonst auch jede Menge lernen (Zeitmanagement, programmiertechnisch, “Politisch”) und das hat uns unglaublich Spass gemacht.

Abbildungsverzeichnis

Tabellenverzeichnis

Versionskontrolle

Version	Datum	Beschreibung	Autor
X0.1	16.12.2017	Dokument erstellt	Dario Carosella, Samuel Ackermann
X0.2	17.12.2017	Kapitel Projektidee & Vorgehensweise erstellt	Dario Carosella, Samuel Ackermann
X0.3	18.12.2017	Dokument überarbeitet (Vorgehensweise)	Dario Carosella, Samuel Ackermann
X0.4	19.12.2017	Kapitel Umsetzung eröffnet	Dario Carosella, Samuel Ackermann
X0.5	20.12.2017	Dokument überarbeitet (Umsetzung)	Dario Carosella, Samuel Ackermann
X0.6	21.12.2017	Kapitel Schlussbericht erstellt	Dario Carosella, Samuel Ackermann
X0.7	22.12.2017	Fertigstellung des Dokumentes	Dario Carosella, Samuel Ackermann
V1.0	22.12.2017	Dokument freigegeben	Dario Carosella