HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

Visualisierung von Service Contracts mit React

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2019

Autoren: Fabio Martins, Jan Hinder
Betreuer: Prof. Dr. Olaf Zimmermann
Projektpartner: Institut für Software, Rapperswil

Abstract

Microservice Domain-Specific Language (MDSL) ist eine domänen-spezifische Sprache zur Spezifikation von Service Contracts (dt. in etwa Schnittstellenverträge) und deren Data Contracts (Daten-Repräsentationen). Die Sprache wurde von unserem Betreuer Prof. Dr. Olaf Zimmermann entwickelt. MDSL abstrahiert von plattformspezifischen Vertragssprachen wie OpenAPI Specification und WSDL sowie integrationsorientierten Programmiersprachen wie Ballerina und Jolie. Die Syntax von MDSL ist vom Domänenmodell der Mustersprache Microservice API Patterns (MAP) inspiriert, die u. a. Endpunkte und Operationen beinhaltet. Bisher gibt es noch keine Visualisierungswerkzeuge für MDSL.

Zu Beginn der Studienarbeit arbeiteten wir ein User-Interface-Konzept aus, das als Basis für die Applikation dient. In diesem überlegten wir uns, wie wir die verschiedenen Sprachelemente von MDSL bestmöglich visualisieren. Die Darstellungen sollten schnell und einfach zu verstehen sein. Zudem sollten die stark vernetzten Sprachbestandteile gut navigierbar sein. Dies erreichen wir, indem wir Visualisierungen wie z. B. die "Hexagonioning"-Grafik oder eine Baumstruktur einsetzen, welche die Sprachelemente übersichtlich wiedergeben.

Im Rahmen unserer Studienarbeit ist ein Visualisierungstool entstanden, in dem eine MDSL-Spezifikation hochgeladen, geparst und angezeigt wird. Die Applikation wurde als serverless Single Page Application (SPA) umgesetzt. Die SPA erlaubt es dem Anwender, in einem agilen Projekt seine Spezifikation schnell zu modellieren und zu erproben. Der Benutzer kann somit jederzeit auf veränderte Kundenanforderungen reagieren. Nach dem Hochladen der MDSL-Spezifikation erscheint eine Statistik mit den Eckdaten der MDSL-Spezifikation. Danach können die verschiedenen MDSL-Sprachelemente wie die Data Contracts, Service Contracts, API Providers und API Clients betrachtet werden. Des Weiteren besteht die Möglichkeit, die MDSL-Spezifikation direkt im Browser zu bearbeiten und herunterzuladen.

Management Summary

Ausgangslage

Die Ausgangslage ist von der Aufgabenstellung gegeben und lautet wie folgt:

«In serviceorientierten Architekturen unterstützen Service Contracts die Interoperabilität zwischen den Service-Providern und ihren Klienten. Diese Verträge müssen sowohl maschinen- als auch menschenlesbar sein. Beispiele für technikspezifische Vertragssprachen sind Open API Specification für RESTful HTTP und WSDL für SOAP-basierte Web Services. Für diese Sprachen existieren zahlreiche Werkzeuge wie Validatoren, Visualisierer und Generatoren. Im Rahmen von Forschungsprojekten sind alternative beziehungsweise ergänzende Sprachvorschläge wie eine Microservices Domain Specific Language (MDSL) [1] entstanden. MDSL exponiert z.B. Microservice API Patterns (MAP) [2] als Sprachkonzepte. Für diese plattformunabhängige, technikneutrale Sprache existieren zwar Editoren und Validatoren sowie erste Generatoren, aber noch kein Visualisierungswerkzeug.» [3]

Vorgehen und Technologie

Zu Beginn der Studienarbeit hatten wir noch keinerlei Erfahrungen im Bereich von domänenspezifischen Sprachen, sowie in Microservices und deren API Patterns. Während der Analyse des Projekts musste genügend Zeit eingeplant werden, um sich mit dem Konzept der Sprache vertraut zu machen. Anhand der Beschreibungen und Beispielen der verschiedenen MDSL-Sprachelemente konnten wir uns schnell einen Überblick über MDSL verschaffen. Danach erarbeiteten wir ein User-Interface-Konzept, das als Basis für unser Visualisierungstool dient. Anschliessend entschieden wir uns für eine Technologie. Die Auswahl fiel auf React, einer Javascript-Bibliothek. React folgt dem Komponentenansatz, dies erlaubt es komplexe Probleme in einfache Komponenten aufzuteilen und führt zu einer besseren Wiederverwendbarkeit sowie Erweiterbarkeit. Diese zwei Faktoren sind Anforderungen im Projekt, die somit abgedeckt werden. Das resultierende Visualisierungstool wurde ohne ein Backend realisiert und ist somit eine serverless Single Page Applikation. Um die hochgeladene MDSL-Spezifikation zu traversieren, wird ANTLR4 verwendet. Mit ANTLR4 konnten wir aus der MDSL-Grammatik Dateien generieren, welche die MDSL-Spezifikation parsen.

Ergebnisse

Entstanden ist ein Visualisierungstool, welches dem Bediener erlaubt, seine MDSL-Spezifikationen nicht nur in textueller Form anzuschauen. Nach dem Hochladen der auszuwertenden Spezifikation wird eine Grafik angezeigt, welche die Eckdaten wiedergibt. Beispielsweise wird die Verteilung der Anzahl der verschiedenen Sprachelemente visualisiert. Der Benutzer hat dann die Möglichkeit alle unterstützten Sprachelemente in dem Tool zu visualisieren. Die Studienarbeit hatte zusätzlich die Anforderung, dass die Applikation für den Beraterалltag tauglich ist. Damit der Berater schnell auf neue Kundenanforderungen reagieren kann, bietet die Applikation die Bearbeitung der MDSL-Spezifikation direkt im Browser an. Die Änderungen werden nach dem Speichern umgehend im Tool übernommen. Die resultierende MDSL-Spezifikation kann anschliessend heruntergeladen werden.

Inhaltsverzeichnis

1 Einleitung und Übersicht

In diesem Dokument ist die Studienarbeit «Visualisierung von Service Contracts mit React» von Fabio Martins und Jan Hinder dokumentiert. Die Arbeit fand im Herbstsemester 2019 an der HSR in Rapperswil statt und wurde von Prof. Dr. Olaf Zimmermann betreut.

1.1 Einführung

In heutigen Applikationen existieren diverse Schnittstellen zwischen Service-Providern und dem Klienten. Diese müssen geregelt werden, was in einem Schnittstellenvertrag definiert wird. Ein Schnittstellenvertrag muss sowohl von einem Computer als auch von einem Menschen interpretiert werden können. Bekannte Vertreter von solchen Vertragssprachen sind Open API Specification für RESTful HTTP und WSDL für SOAP-basierte Web Services. Für diese Vertragssprachen gibt es bereits diverse Tools zur Visualisierung. Allerdings unterstützen diese Sprachen nur ein spezielles Protokoll.

In einem Forschungsprojekt entstand die Sprache Microservices Domain Specific Language (MDSL). [1] MDSL ist plattformunabhängig und technikneutral, somit unterstützt sie diverse Protokolle zur Übertragung. Des Weiteren implementiert MDSL die Microservice API Patterns (MAP) [2] als Sprachkonzepte. Erste Editoren und Generatoren gibt es bereits, allerdings existiert für MDSL noch kein Visualisierungstool. Bisher gibt es nur die textuelle Ansicht in einem Editor.

In dieser Arbeit wurde ein Visualisierungstool für MDSL-Spezifikationen entworfen und entwickelt. Im Tool kann eine MDSL-Spezifikation hochgeladen werden. Diese wird vom Tool aufbereitet und schlussendlich grafisch und textuell im Browser visualisiert. Das Visualisierungstool zeigt zum einen die verschiedenen MDSL-Sprachelemente an, anderseits erscheint nach dem Hochladen der Spezifikation auch eine Metrik mit den wichtigsten Eigenschaften. Weiter besteht die Möglichkeit die Spezifikation zu editieren. Dies ermöglicht es die Auswirkungen von Änderungen gleich im Browser festzustellen. Die geänderte MDSL-Spezifikation kann schlussendlich auch heruntergeladen und gespeichert werden.

Mit Hilfe des Visualisierungstool sollen sowohl erfahrene Service Designer als auch Gelegenheitsnutzer ihre MDSL-Spezifikationen schnell und einfach visualisieren lassen.

1.2 Struktur des Dokuments

Das Dokument ist in sechs Kapitel unterteilt. Das aktuelle Kapitel beinhaltet eine kurze Einleitung in unsere Studienarbeit. Im Kapitel «Anforderungen» werden hauptsätzlich die funktionalen sowie die nicht-funktionalen Anforderungen beschrieben. Darauf folgt das Kapitel «Architektur», in diesem wird die Architektur unserer Applikation aufgezeigt. Weiter geht es mit dem UI-Konzept, wo wir das geplante User-Interface aufzeigen. Im Kapitel «Umsetzung» beschreiben wir einerseits alle Library- und Toolentscheide, die wir während des Projektes getroffen haben. Andererseits ist in diesem Kapitel die effektive Implementierung und das Testing dokumentiert. Im letzten Kapitel reflektieren wir das Projekt. Somit werden in diesem Kapitel die Ergebnisse beschrieben und Vergleich mit den Anforderungen durchgeführt. Zusätzlich wird ein Ausblick in die Erweiterbarkeit der Applikation gegeben.

2 Anforderungen

In der Anforderungsanalyse haben wir die allgemeine Beschreibung sowie die funktionalen und nicht-funktionalen Anforderungen definiert.

2.1 Allgemeine Beschreibung

Zuerst definieren wir die kurz die allgemeine Beschreibung der Applikation.

2.1.1 Produktperspektive

Es soll eine Visualisierung von Service und Data Contracts erstellt werden. Dies soll dem Benutzer helfen eine Datei besser und einfacher zu verstehen, anstatt nur die Textansicht zu sehen. Dazu lädt ein Benutzer in der Applikation eine MDSL-Spezifikation hoch, die analysiert und visualisiert wird. Die Applikation ist eine Web-Anwendung, wodurch eine Datei direkt im Browser analysiert wird.

2.1.2 Benutzermerkmale

Die Zielgruppe von dieser Visualisierung besteht vorwiegend aus erfahrenen Service Designern sowie aus Gelegenheitsnutzern, welche Web APIs und Service Contracts schnell modellieren und erproben können.

2.1.3 Einschränkungen

Es gibt in diesem Projekt einige Einschränkungen, die hier kurz erläutert werden.

Da die Anwendung eine Web-Applikation ist, kann sie nur bei einer aktiven Internetverbindung verwendet werden. Andernfalls muss die Applikation lokal gestartet werden.

Es wurde nicht auf spezielle Browser-Anforderungen oder Versionen geachtet. Die App wurde mit den aktuellen Versionen von Google Chrome und Mozilla Firefox implementiert und getestet. Die Kompatibilität mit älteren Versionen, vor allem in Bezug auf Internet Explorer, wird nicht gewährleistet.

2.1.4 Abhängigkeiten

Da die Applikation als Web-Applikation zugänglich ist, besteht die Abhängigkeit bei den Benutzern nur darin, dass sie einen aktuellen Web Browser installiert haben müssen.

Weiter gibt es vor allem im Frontend-Bereich Abhängigkeiten von Bibliotheken, die verwendet werden, um die Realisierung der Applikation zu vereinfachen. Diese Bibliotheken wurden deswegen als Kopie eingebunden, um nicht von allfälligen Änderungen betroffen zu sein.

2.2 Funktionale Anforderungen

Die funktionalen Anforderungen haben wir als User-Stories beschrieben.

Als Service Designer möchte ich MDSL-Spezifikationen in die Webanwendung hochladen können. Die in der Spezifikation enthaltenen Sprachelemente sollen dann von der Anwendung graphisch und textuell visualisiert werden. Somit ist diese visuelle MDSL-Spezifikation schneller und einfacher zu verstehen als eine textuelle. [1]

Die folgende User-Story ist optional.

Als Service Designer möchte ich CML-Spezifikationen in die Webanwendung hochladen können. Die in der Spezifikation enthaltenen Sprachelemente sollen dann von der Anwendung graphisch und textuell visualisiert werden. Somit ist diese visuelle CML-Spezifikation schneller und einfacher zu verstehen als eine textuelle. [4]

```
API description SampleCustomerManagementAPI
usage context PUBLIC_API for FRONTEND_INTEGRATION

endpoint type CustomerManagementContract
  version 1.0.0
  serves as INFORMATION_HOLDER_RESOURCE
  exposes
    operation lookupSingleCustomer
      with responsibility RETRIEVAL_OPERATION
      expecting
        payload ID<string>
      delivering
        payload {"customerId":ID<int>,
                 "name":V,
                 "address"}

    operation lookupCustomerDirectory
      with responsibility RETRIEVAL_OPERATION
      expecting
        payload <<Request_Bundle>> "customerId":ID<int>+
      delivering
        payload
          "customerRecord": {
            "cid":ID!, // ! mandatory, exactly one
            "nameTuple":("firstname":V, "lastname":V),
            "addressTuple":(
              "street":V<string>,
              "poBox":V?, // optional
              "zipCode":V,
              "city":V)+,
            "segment":("REGULAR":V|"VIP":V) // choice
          }* // zero or more
```

Abbildung 1: Sprachbeispiel eines MDSL Service Contracts [5]

2.3 Nicht-funktionale Anforderungen

Für unsere Applikation haben wir folgende nicht-funktionalen Anforderungen definiert.

2.3.1 Performance

Eine MDSL-Spezifikation soll nach dem Hochladen in minimal einer Sekunde, optimal fünf Sekunden und maximal sieben Sekunden im Webbrowser dargestellt sein.

2.3.2 Suitability

Für erfahrene Service Designer muss es möglich sein, sich in ein bis zwei Stunden in das Tool einarbeiten zu können. Das Tool kann für die Durchführung von Design Workshops eingesetzt werden.

2.3.3 Accuracy

Der Benutzer kann nur gültige MDSL-Spezifikationen in die Webanwendung hochladen.

2.3.4 Reliability

Unser Betreuer ist auch Berater und testet, ob die Webanwendung im Beraterалltag eingesetzt werden kann. Es wird ein Usability Test mit dem Betreuer durchgeführt. Es dürfen keine Daten verloren gehen.

2.3.5 Transferability

Die Webapplikation kann mit der Installationsanleitung in minimal einer Minute, optimal fünf Minuten und maximal 15 Minuten installiert werden.

2.3.6 Usability

Der Einstieg in den Webauftritt für die Visualisierung von Service Contracts soll für den User möglichst simpel und intuitiv gestaltet werden. Die Applikation muss so erstellt werden, dass ein erfahrener Service Designer das Konzept des Systems innerhalb von maximal einer halben Stunde erlernen und verstehen kann. Ein Domänen Fachexperte ohne Erfahrung in SOA oder Microservices soll nach Einarbeitung von maximal zwei Stunden Service Endpoints und Data Contracts visualisieren können.

2.3.7 Maintainability

Mit React- und MDSL-Erfahrungen können neue Notationen und MDSL-Konzepte innerhalb von maximal einem Tag implementiert werden.

2.3.8 Portability

Die Applikation kann mit Smartphones, Tablets und Laptops aufgerufen werden und soll dementsprechend responsive sein.

2.3.9 Compatibility (optional)

Die Visualisierung der MDSL Service oder Data Contracts sind in andere Webseiten einbettbar.

2.3.10 User Interface Begrenzungen

Unter diesem Punkt legen wir für alle 1:n oder n:m Beziehungen der MDSL Grammatik fest, wie viele Elemente im User Interface jeweils schön dargestellt werden sollen.

2.3.10.1 serviceSpecification

Da Data Contracts, Service Contracts, Clients und Provider essenziell für eine MDSL-Spezifikation sind, werden alle deklarierten Sprachelemente in der Spezifikation im User Interface angezeigt.

2.3.10.2 Weitere Sprachelemente

Folgend ist eine Auflistung der weiteren Sprachelemente.

Sprachelement	garantiert darzustellende Sprachelemente
otherRoles	5
operations	10
Data Contracts Verschachtelungstiefe	5
Data Contracts Verschachtelungstiefe in Operation	2
endpointList	5
endpointInstance	3
moreNames	5
consumption	10

Tabelle 1: User Interface Begrenzungen

2.4 Umzusetzende Sprachelemente

In der folgenden Tabelle sind alle Sprachelemente aufgelistet mit ihrer Priorität. Diese wurden während der genauen Definition der Aufgabenstellung mit unserem Betreuer festgelegt.

Sprachelement	high	medium	ignoring
MDSL Grammar Quick Reference (Skeletons) [6] Service Contract Skeleton	X		
MDSL Grammar Quick Reference (Skeletons) Reporting		X	
Service Endpoint Contracts in MDSL [5] Concepts	X		
Service Endpoint Contracts in MDSL Concepts (identified by)		X	
Service Endpoint Contracts in MDSL Security Policy		X	
Service Endpoint Contracts in MDSL Parameterbinding		X	
Data Contracts and Schemas in MDSL [7]	X		
Runtime Language Concepts: API Provider, API Client, API Gateway [8] API Provider	X		
Runtime Language Concepts: API Provider, API Client, API Gateway API Provider - SLA		X	
Runtime Language Concepts: API Provider, API Client, API Gateway API Client	X		
Runtime Language Concepts: API Provider, API Client, API Gateway API Gateway		X	
Runtime Language Concepts: API Provider, API Client, API Gateway Service Composition			X
MDSL.xText ab Zeile 377			X

Tabelle 2: Umzusetzende Sprachelemente

3 Architektur

Im diesem Kapitel behandeln wir die Systemübersicht, die logische Architektur und das Deployment.

3.1 Systemübersicht

Die Softwarearchitektur besteht aus einer Serverless Single-Page Application (SPA). Die Bestandteile des Systems werden nachfolgend beschrieben.

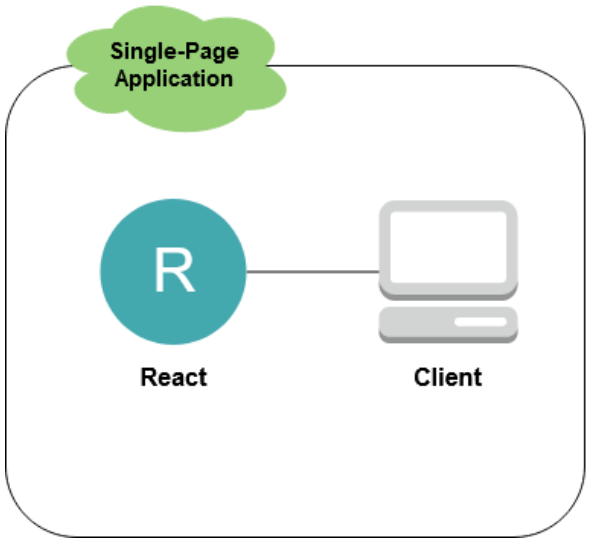

Abbildung 2: Systemübersicht

3.1.1 React-Applikation

Im Frontend verwenden wir die JavaScript-Library React als Grundgerüst. [9] Mit dieser Library bieten wir eine performante SPA an, mit welcher die Benutzer MDSL-Spezifikationen visualisieren können. [1] Der User führt die alleinstehende SPA auf seinem Computer aus.

3.1.2 Components

In React können Komponenten als Klassen oder als Funktionen definiert werden. Die Verwendung von Klassen bietet derzeit mehr Funktionen an. Jede Klasse besitzt mehrere Lebenszyklusmethoden, die überschrieben werden können, um Code zu bestimmten Zeiten im Prozess aufzurufen. Falls ein State oder eine Lebenszyklusmethode nötig ist, arbeiten wir mit der ES6-Klassen Syntax, ansonsten mit Funktionen. Die Struktur mit den Komponenten erlaubt es, die Benutzeroberfläche in unabhängige wiederverwendbare Teile aufzubrechen und jede Komponente als isoliert zu betrachten. [10]

3.1.2.1 ECMAScript 6

ECMAScript 6 (ES6) ist ein Standard, der die offizielle Spezifikation von Javascript ist. In ES6 gab es in Javascript erstmals die Option Klassen zu verwenden, wie man es von Java oder C# kennt. [11]

3.2 Architektonische Ziele & Einschränkungen

In diesem Kapitel werden Architekturentscheide beschrieben, um die nicht-funktionalen Anforderungen zu erreichen.

3.2.1 Performance

Um eine bessere Performance zu erzielen, haben wir den Parser der MDSL-Spezifikation direkt im Frontend implementiert. Somit geschieht das Parsing direkt beim Client, ohne dass ein Request an den Server gesendet wird.

3.2.2 Transferability

Damit die Applikation schnell installiert und gestartet werden kann, läuft die Applikation in einem Docker-Container. Dadurch kann der Docker-Container und somit die Applikation schnell und einfach gestartet werden. Zudem kann dieser auf jedem System gestartet werden, auf dem Docker installiert ist. Die Applikation kann allerdings auch via npm ausgeführt werden. [12]

3.2.3 Maintainability

Das Frontend wurde möglichst modular entwickelt, damit die einzelnen Komponenten wiederverwendbar sind. Somit sollen neue Notationen und MDSL-Konzepte schnell implementiert werden können.

3.3 Logische Architektur

Folgend ist beschrieben, wie der Presentation-Layer strukturiert ist. Zudem werden die einzelnen Komponenten beschrieben.

3.3.1 Presentation

Der Presentation-Layer ist als React-SPA aufgebaut. Im unteren Diagramm ist die Struktur zu sehen, wie unsere Applikation aufgebaut ist.

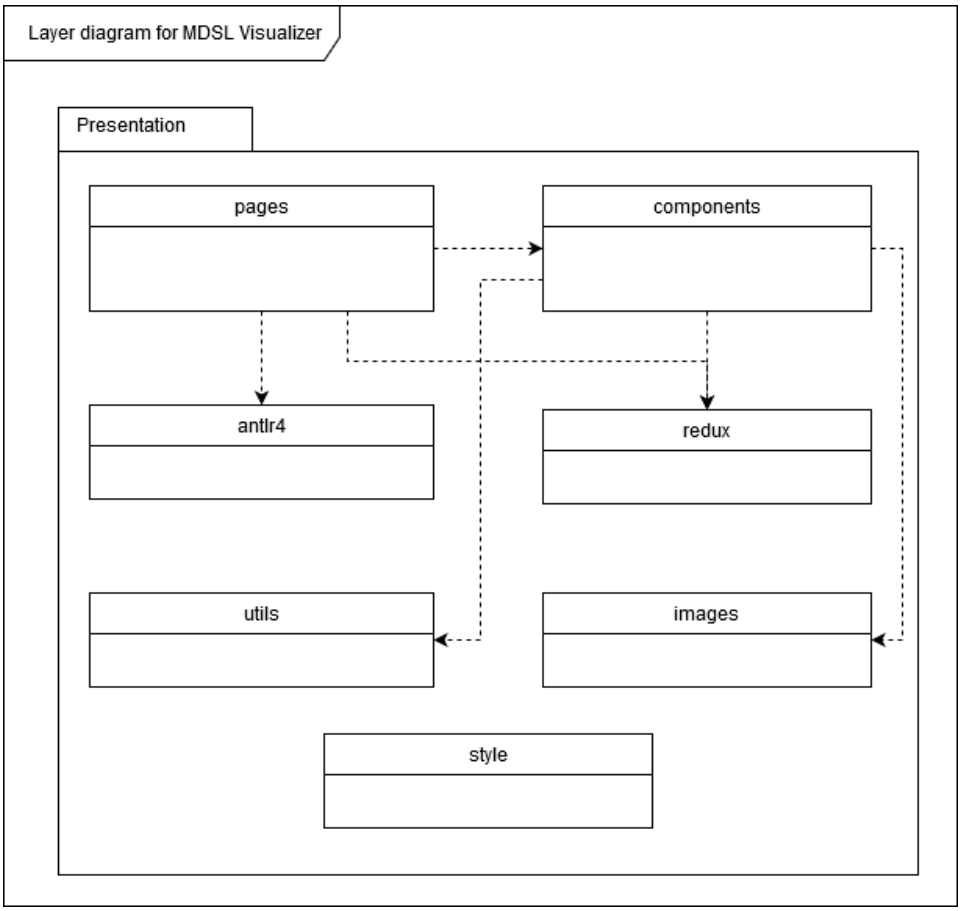

Abbildung 3: Schichtendiagramm des MDSL Visualizers

3.3.2 Beschreibung der Komponenten des Schichtendiagramms

Hier werden die einzelnen Komponenten genauer beschrieben.

3.3.2.1 pages

Unter pages sind React-Komponenten zu finden, die eine Seite in der Applikation repräsentieren. Für die MDSL-Sprachelemente Client, Data Contract, Provider und Service Endpoint gibt es jeweils eine eigene Seite. Zusätzlich kommen noch die MainPage und die Overview hinzu, die für den Datei-Upload respektive für die Anzeige der API Description und Übersicht zuständig sind. Die genaue Beschreibung all dieser React-Komponenten inklusive Abhängigkeiten sind im UI-Konzept zu finden.

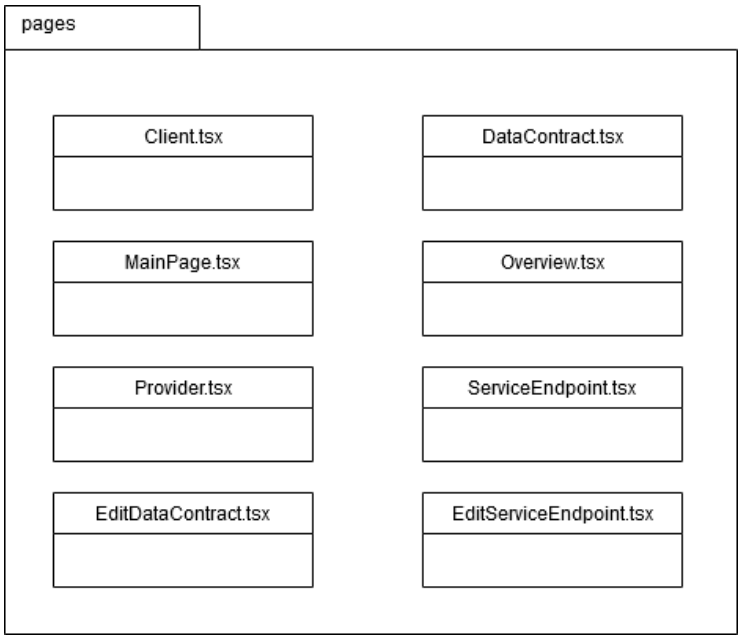

Abbildung 4: Komponentendiagramm der pages

EditDataContract und EditServiceEndpoint

Weil die Platzverhältnisse auf den entsprechenden Seiten knapp waren, haben wir diese zwei Komponenten in separate Seiten gepackt. Diese bieten lediglich die Funktionalität an, den Data Contract oder den Service Endpoint zu bearbeiten und zu speichern.

3.3.2.2 components

Bei den components sind die einzelnen React-Komponenten zu finden. Hier gibt es wieder Komponenten für die MDSL-Sprachelemente sowie für die MainPage und die Overview. Diese beinhalten diverse kleinere Unterkomponenten für die jeweilige Seite. Somit konnten wir die Komponenten klein halten.

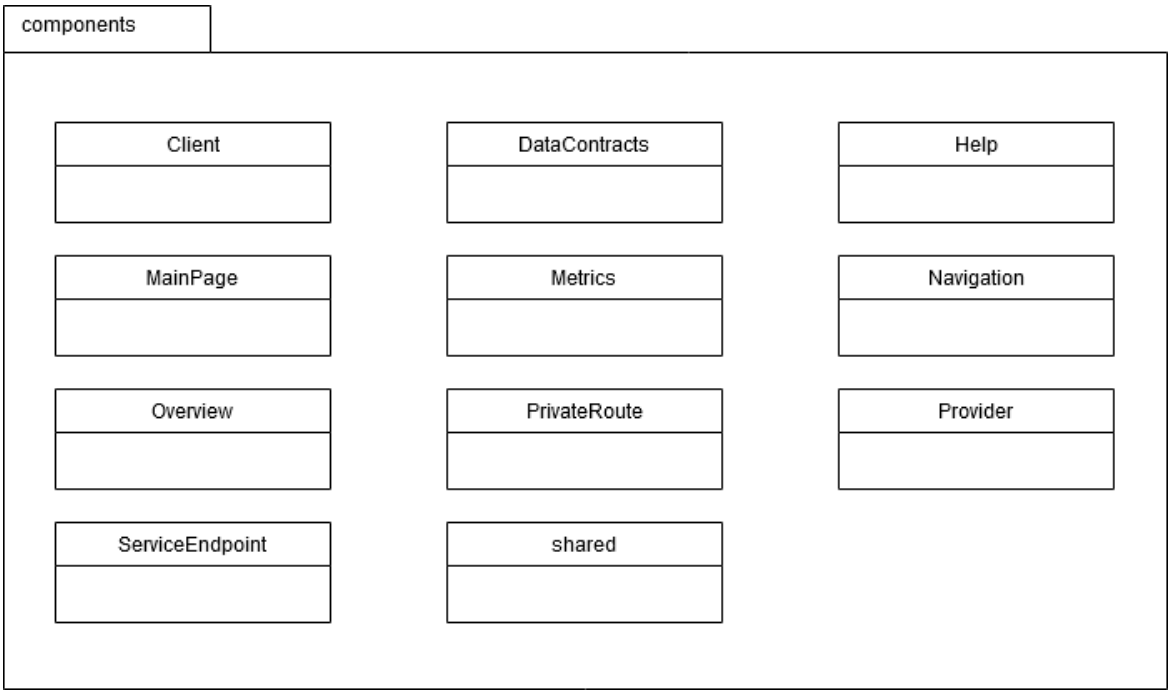

Abbildung 5: Komponentendiagramm der components

Help

In der Navigation existiert ein Help-Link, der die Help-Komponente öffnet. Diese erscheint als Popup und zeigt eine kurze Beschreibung zu MDSL und den MDSL-Sprachelementen an. Help kann von allen Seiten aufgerufen werden.

Metrics

Um die Eckdaten nach dem Hochladen und Parsen einer MDSL-Spezifikation darzustellen, wird die Metrics-Komponente benötigt. Die relevanten Daten werden aus dem Parse Tree entnommen. Die Metriken sind nur auf der Hauptseite ersichtlich.

Navigation

Die Navigation beinhaltet die Menubar, die Breadcrumb-Navigation und den Footer. Die Menubar und der Footer sind direkt in der Root-Komponente eingebunden. Die Breadcrumb-Navigation ist bei jeder Seite eingebunden.

PrivateRoute

Damit bei ungültigen Link-Aufrufen oder bei Nichtvorhandensein des Parse Trees keine Fehlermeldung erscheint, gibt es die PrivateRoute. Wenn eine der Bedingungen nicht erfüllt ist, wird der Benutzer auf die Hauptseite weitergeleitet. Somit sind alle Routen bis auf die Hauptseite als PrivateRouten deklariert.

shared

Hier sind die React-Komponenten, die mehrfach verwendet werden. So gibt es beispielsweise den PatternMapper, der prüft, ob das Pattern existiert und es ein Icon für das Pattern gibt. Schlussendlich retourniert dieser den Patternnamen als Link auf die MAP-Seite und falls vorhanden das Icon. [2]

3.3.2.3 antlr4

Unter generatedAntlr4Files sind die generierten Dateien vom ANTLR Tool zu finden, wie der Parser und der Lexer. [13] Weil diese Dateien generiert wurden, wird die Funktionalität nicht erläutert.

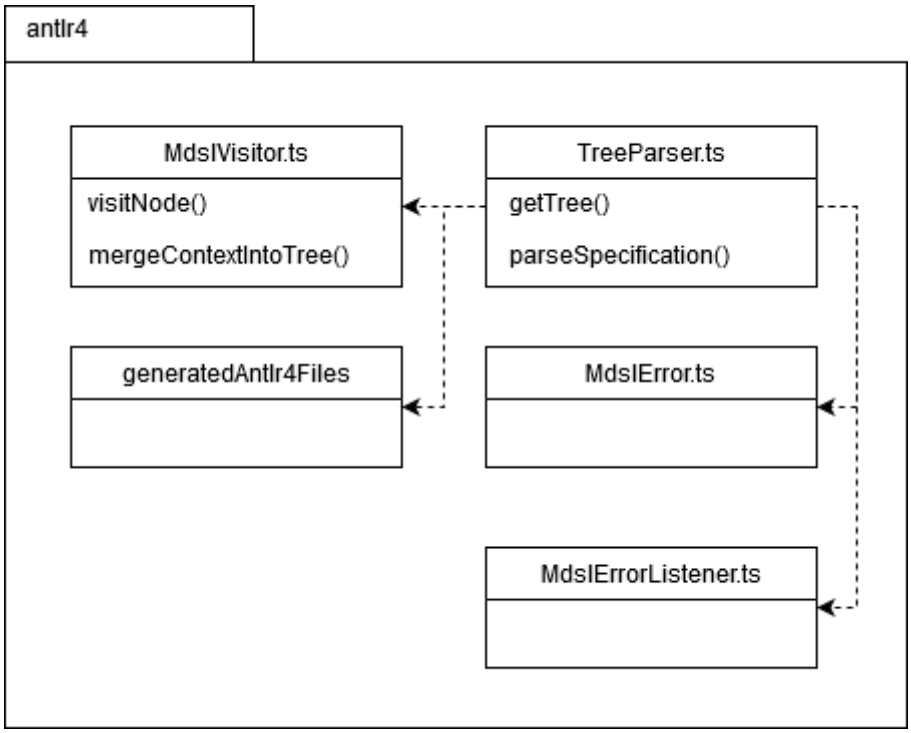

Abbildung 6: Komponentendiagramm von antlr4

MdslVisitor

Der MdslVisitor erhält den Parse Tree und bereitet die Daten bei der Traversierung des Baumes auf. Wie es der Name bereits sagt, implementiert der MdslVisitor das Visitor-Pattern. [14] Dabei ruft er rekursiv seine Kinder auf. Dieser wird einzig vom TreeParser aufgerufen.

TreeParser

Im TreeParser wird die MDSL-Spezifikation geparst. Daraus resultiert der Parse Tree, der später an den MdslVisitor weitergegeben wird.

MdslError

Um ungültige MDSL-Spezifikationen handzuhaben, wird bei einem Fehler ein MdslError erstellt. Dabei wird die Zeile, die Spalte sowie die Fehlermeldung gespeichert. Somit kann die Fehlermeldung auf der Hauptseite angezeigt werden.

MdslErrorListener

Damit die Fehler beim Parsen der MDSL-Spezifikation gespeichert werden, benötigt es einen Error Listener. Der MdslErrorListener ist somit beim Lexer sowie beim Parser registriert.

3.3.2.4 redux

Redux ist eine Library für das Statemanagement in React. Redux ermöglicht es, dass mehrere React-Komponenten auf denselben State zugreifen können. Da Redux nicht standardmässig bei React installiert ist, haben wir uns entschieden, unsere Redux-Struktur hier aufzuzeigen und kurz zu erklären. [9]

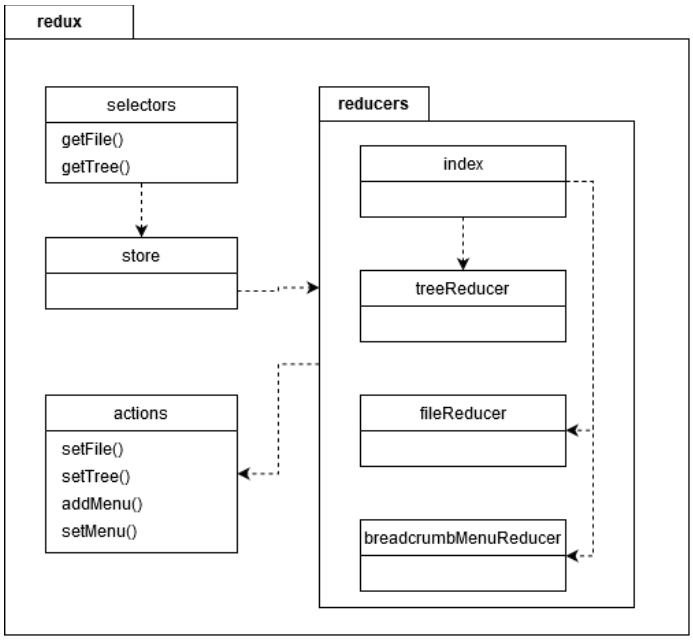

Abbildung 7: Komponentendiagramm von Redux

selectors

Ein Selector gibt das gewünschte Element aus dem Store zurück.

store

Der Store beinhaltet den State. Dies ist ein Tree von Objekten.

actions

Um eine Stateänderung im Store zu vollziehen, benötigt es Actions. Diese werden von den Komponenten aufgerufen.

reducers

Ein Reducer erhält eine Action. Anhand der Action und des aktuellen States, erstellt der Reducer einen neuen State. Ein Reducer ist allerdings immer nur für einen Teil des States zuständig.

3.3.2.5 utils

Unter utils sind zwei Hilfsklassen zu finden. Diese beinhalten jeweils mehrere Funktionen, die von den Komponenten mehrfach verwendet werden.

3.3.2.6 images

Hier sind alle MAP-Icons zu finden, die in der MDSL-Grammatik vorkommen.

3.3.2.7 style

In diesem Package sind alle Dateien abgelegt, die dem Stylen der Anwendung dienen.

3.4 Deployment

Während der Studienarbeit ist die Applikation auf dem zur Verfügung gestelltem Server erreichbar. Damit die Applikation später auf einem anderen Server ausgeführt werden kann, setzen wir Docker-Container ein.

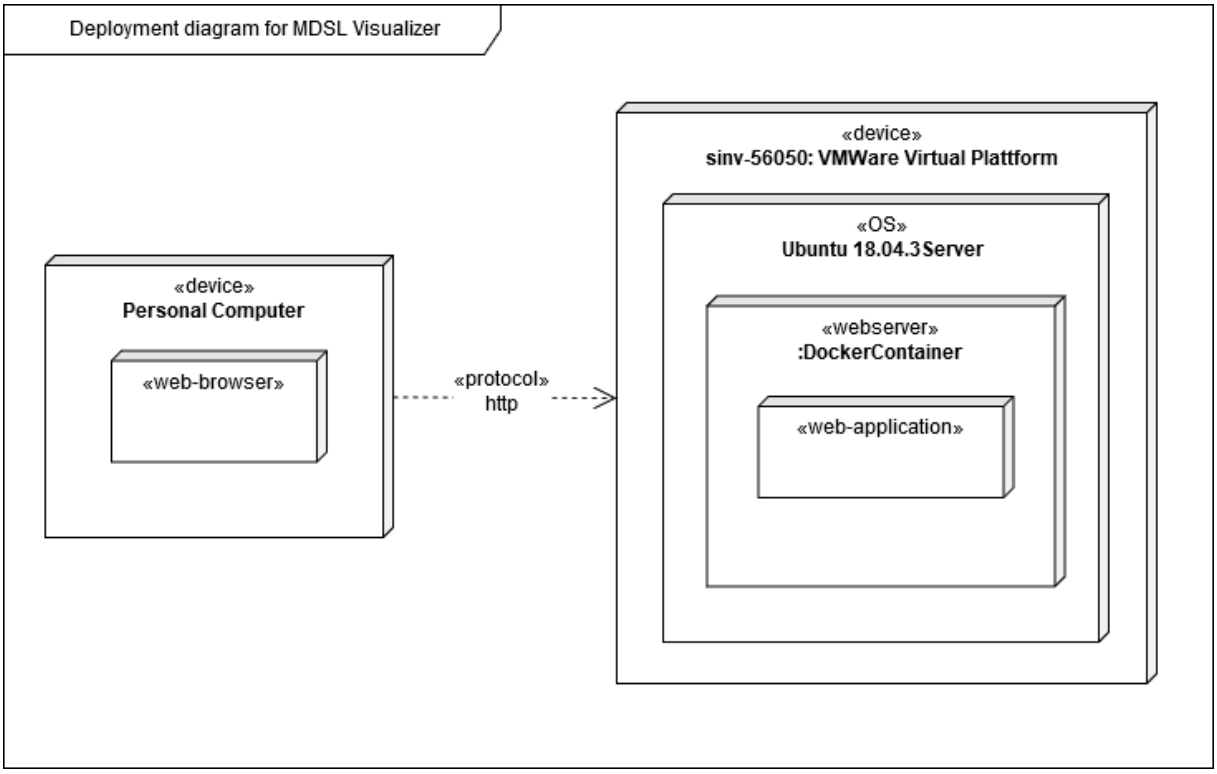

Abbildung 8: Deployment-Diagramm

3.4.1 Beschreibung der Komponenten

Folgend sind die Komponenten des Deployment-Diagramms beschrieben.

3.4.1.1 Web-Application

Die Web-Application stellt die eigentliche Applikation zur Verfügung. Diese läuft in einem Docker-Container und wird in einer Node-Umgebung ausgeführt. Im Docker-Container läuft diese auf dem Port 3000.

3.4.1.2 Docker Container

Den Webserver starten wir als Docker-Container, womit der Server nur Docker unterstützen muss, um die Applikation auszuführen. [15] Weiter kann die Applikation somit ohne grossen Konfigurationsaufwand auf einem anderen Server zur Verfügung gestellt werden. Im Docker-Container läuft eine Node-Applikation. Der Docker-Container selbst läuft auf dem Port 80, der veränderbar ist. Somit ist die Applikation schlussendlich auf diesem Port erreichbar.

3.4.1.3 Personal Computer

Der Personal Computer (PC) stellt einen Computer eines Benutzers der Web-Applikation dar. Der PC lädt die Applikation via HTTP vom Server herunter und führt diese im Webbrowser aus.

4 User-Interface-Konzept

In nächsten Unterkapiteln sind unsere User-Interface Skizzen für unsere Applikation. Diese haben wir nach den MDSL-Sprachelementen aufgeteilt. Die Anzahl an zu unterstützenden Elementen sind in den Anforderungen unter Punkt 2.3.10 definiert. [1]

4.1 Inspiration

Für die Visualisierung der API Clients und API Provider verwenden wir die «hexagonioning»-Darstellung. [16] [17] Für die restlichen Elemente haben wir anhand der MDSL-Grammatik selbst Visualisierungsmöglichkeiten entworfen. Um ein anschauliches UI zu konzipieren, werden wir auf den Lerninhalt des bereits besuchten HSR-Moduls «Human Centered Interaction Design» zurückgreifen.

4.2 Navigation

Damit der User sicherlich zu jeder Zeit weiss, wo er sich gerade befindet, werden wir eine Navigation mittels Breadcrumb einsetzen. Der Anwender kann somit beliebig in der MDSL-Spezifikation navigieren. Von React-Bootstrap gibt es bereits ein Beispiel für die Verwendung von Breadcrumb. [18]

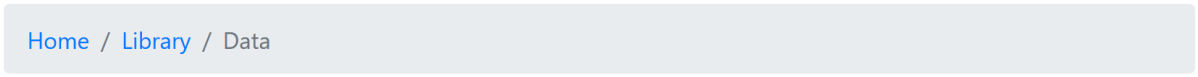

Abbildung 9: React-Bootstrap Breadcrumb

4.3 Hauptseite

Beim Öffnen der Web-Applikation erscheint zuerst die Hauptseite. Auf dieser lädt der User eine MDSL-Spezifikation hoch und wird folglich auf eine Übersicht der Spezifikation weitergeleitet.

4.3.1 Visualisierung

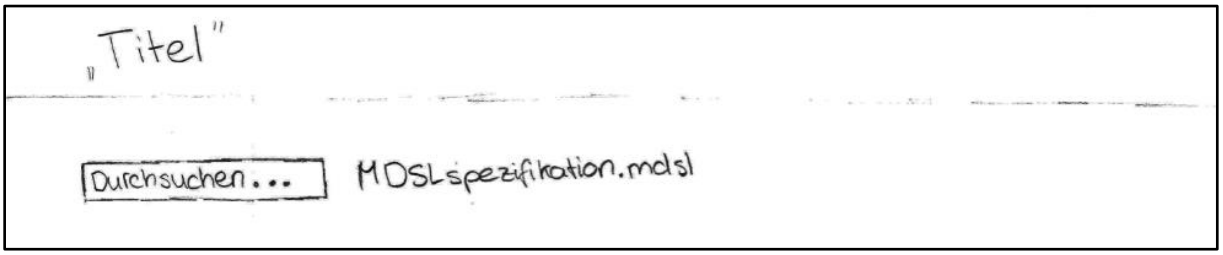

Abbildung 10: Visualisierung der Hauptseite

4.3.2 Beschreibung

Die Hauptseite ist in zwei Segmente unterteilt. Im oberen Bereich sieht der User den Titel der Applikation und kann gleich unterhalb des Titels eine MDSL-Spezifikation angeben. Dies erfolgt durch einen File-Input der entsprechend mit Bootstrap gestylt wird. Beim Input-Button sieht man auch gleich die ausgewählte Spezifikation.

4.4 Overview

Folgend ist die Ansicht der Overview beschrieben.

4.4.1 Code Beispiel [1]

```
API description HelloWorldAPI
  usage context PUBLIC_API for FRONTEND_INTEGRATION

data type SampleDTO {ID, V}

endpoint type HelloWorldEndpoint
exposes
  operation sayHello
    expecting payload V<string>
    delivering payload SampleDTO

API provider HelloWorldAPIProvider1
  offers HelloWorldEndpoint

API client HelloWorldAPIClient1
  consumes HelloWorldEndpoint
```

4.4.2 Visualisierung

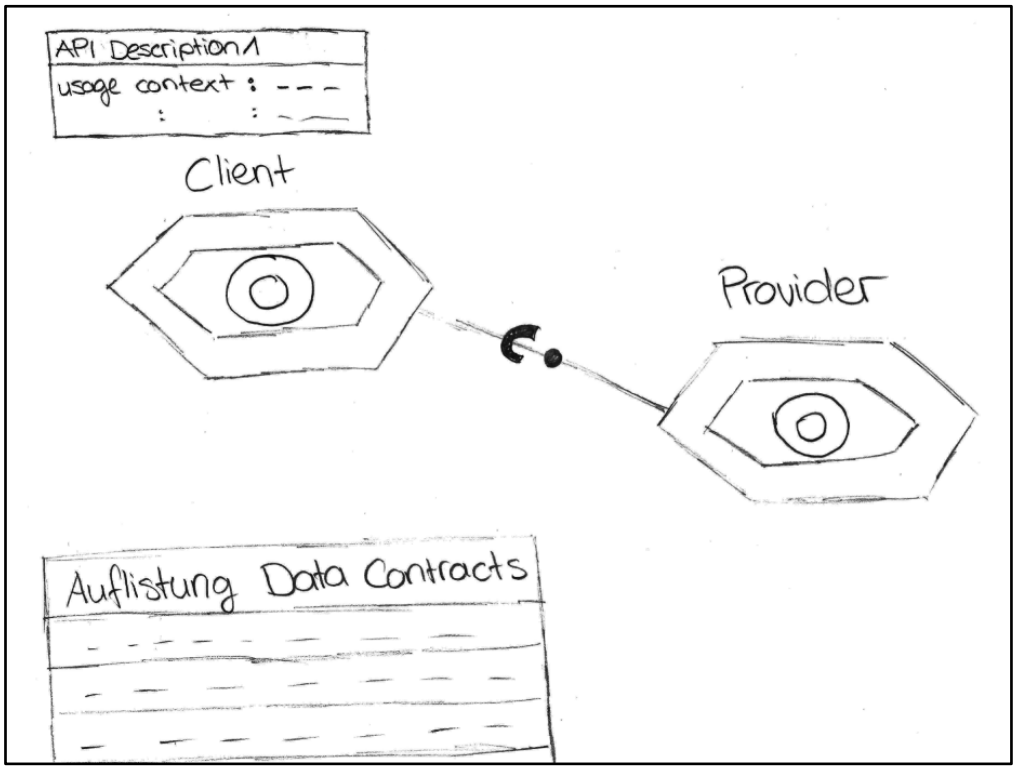

Abbildung 11: Visualisierung der Overview

4.4.3 Beschreibung

Die Overview ist in drei Segmente unterteilt. Oben sieht man die Beschreibung der Spezifikation. Im mittleren Bereich sind API Client und API Provider ersichtlich, die mittels einer hexagonialen Grafik visualisiert werden. [1] Die Hexagons sind mit einer Beziehung untereinander verbunden. Im unteren Bereich werden alle erkannten Data Contracts aufgelistet.

4.4.4 Abhängigkeiten

Beim Klick auf das linke Hexagon kommt der Anwender zu einer neuen Seite, in der die einzelnen API Clients angezeigt werden. Klickt der Benutzer auf das rechte Hexagon, öffnet sich die neue Seite mit keinem oder mehreren API Providern. Bei der Auswahl eines Data Contracts in der Liste öffnet sich die Detailansicht zu diesem spezifischen Data Contract.

4.4.5 Begründung

Wir haben uns für den Einsatz von zwei Hexagons entschieden, weil diese Visualisierung von der Microservice Community empfohlen wird. Somit setzen wir auf eine bereits bekannte Visualisierung in der Microservice Umgebung. Um einen schnellen Überblick über alle Data Contracts zu erhalten, setzen wir eine Tabelle ein.

4.5 Client

Folgend ist die Ansicht von den API Clients beschrieben.

4.5.1 Code Beispiel [8]

```
API client SampleAPIClient
  consumes contractTemplate
  from providerTemplate
  via protocol RESTful_HTTP
```

4.5.2 Visualisierung

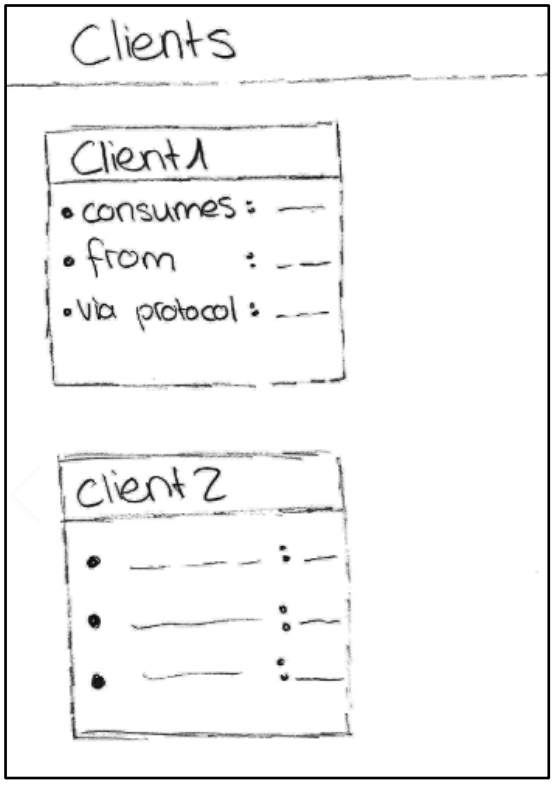

Abbildung 12: Visualisierung der Clients

4.5.3 Beschreibung

Durch die Startseite gelingt der User auf die Ansicht mit den einzelnen API Clients. Ein Client konsumiert einen oder mehrere Endpoint Contracts. Bei «from» wird auf den Provider referenziert. Der User sieht, welches Protokoll verwendet wird. Für die Visualisierung verwenden wir Cards von Bootstrap. [19] Die Details sind lediglich in textueller Form ersichtlich.

4.5.4 Abhängigkeiten

Die einzelnen Endpoint Contracts unter «consumes» werden Links sein. Durch den Klick auf diese Links kommt der Bediener auf die entsprechenden Endpoint Contracts. Die Provider sind ebenfalls mit Links erreichbar.

4.5.5 Begründung

Für die Kartenansicht haben wir uns entschieden, weil es auf einem Blick erkenntlich ist, welche Informationen zu welchem Client gehören. Somit wird bereits durch den Einsatz dieses visuellen Elements eine Gruppierung durchgeführt. Dem User fällt es dadurch einfacher, die einzelnen Clients zu unterscheiden.

4.6 Provider

Folgend ist die Ansicht eines API Providers beschrieben.

4.6.1 Code Beispiel [8]

```
API provider sampleProviderTemplate
  offers sampleContractTemplate
    at endpoint location http://www.tbc.io:80/path/subpath
    via protocol RESTful_HTTP
```

4.6.2 Visualisierung

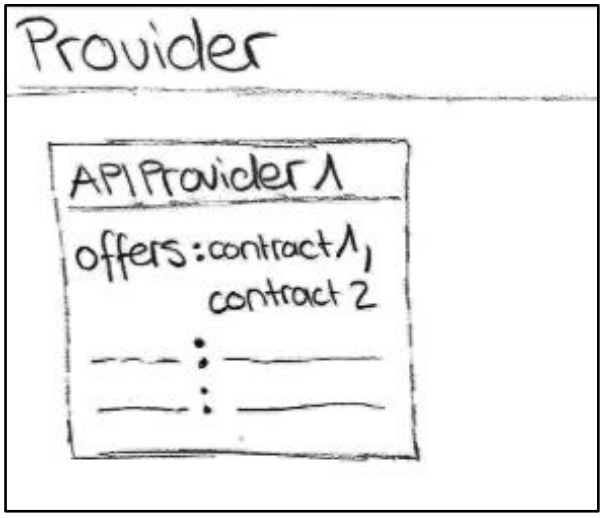

Abbildung 13: Visualisierung des Providers

4.6.3 Beschreibung

Über die Startseite navigiert der User auf die API Provider. Ein API Provider stellt einen oder mehrere Service Endpoint Contracts an einer Adresse zur Verfügung. Für die Visualisierung von API Providern wird die Cardview von Bootstrap verwendet. [19]

4.6.4 Abhängigkeiten

Die einzelnen Contracts unter «offers» sind Links. Durch diese öffnet sich der spezifische Endpoint Contract. Falls notwendig wird bei den anderen Punkten ebenfalls mit Links gearbeitet.

4.6.5 Begründung

Aus demselben Grund wie beim Client, verwenden wir beim Provider die Kartenansicht. Ebenfalls wollen wir, wenn möglich, nicht zu viele verschiedene visuelle Elemente einfügen. Der Benutzer soll keine unnötigen Visualisierungen zu sehen bekommen.

4.7 Service Contract

Folgend ist die Ansicht eines einzelnen Service Endpoints beschrieben.

4.7.1 Code Beispiel [5]

```
endpoint type CustomerManagementContract
  version 1.0.0
  serves as INFORMATION_HOLDER_RESOURCE
  exposes
       operation lookupSingleCustomer
         with responsibility RETRIEVAL_OPERATION
         expecting
              payload ID<string>
         delivering
              payload {"customerId":ID<int>, "name":V, "address"}

       operation lookupCustomerDirectory
         with responsibility RETRIEVAL_OPERATION
         expecting
              payload <<Request_Bundle>> "customerId":ID<int>+
         delivering
              payload CustomerRecord
```

4.7.2 Visualisierung

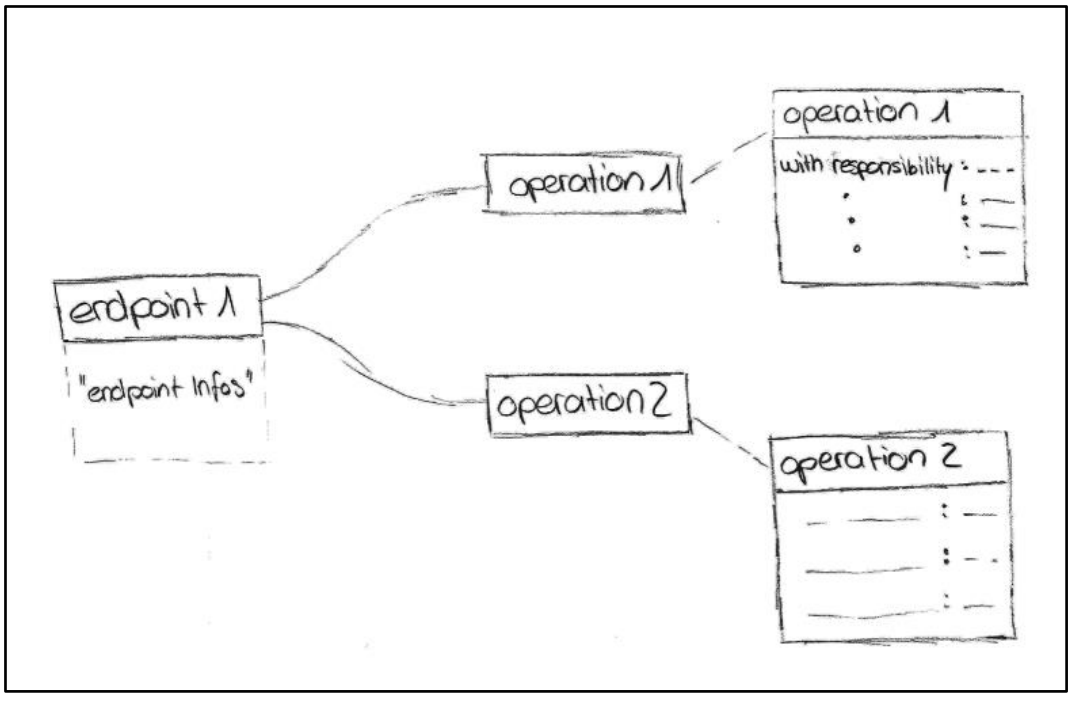

Abbildung 14: Visualisierung der Service Endpoints

4.7.3 Beschreibung

Ein Service Endpoint mit all seinen Operationen wird als Baum dargestellt. Beim Klick auf den Endpoint werden somit alle Operationen dieses Service Endpoints angezeigt. Weiter öffnet sich beim Klick auf eine Operation auf der rechten Seite eine Detailansicht dieser Operation. Bei einer Operation kann unterschieden werden, ob es sich dabei um einen Request oder Request-Reply handelt. Dieser Unterschied einer Operation wird anhand einer Grafik visualisiert.

4.7.4 Abhängigkeiten

Bei den Details einer Operation sind die verwendeten Data Contracts zu sehen. Zusätzlich lässt sich eine Detailansicht von den Data Contracts öffnen, sofern der Data Contract nicht inline definiert ist. Falls in der Operation ein Microservice API Pattern verwendet wird, wird das Icon von diesem Pattern angezeigt und auf die entsprechende MAP-Webseite verlinkt. [2]

4.7.5 Begründung

Da der Aufbau eines Services Contracts einer Baumstruktur entspricht, visualisieren wir den Service Contract mit seinen Operationen als Baum. Mit der Request oder Request-Reply Grafik wollen wir bezwecken, dass der User auf einem Blick sieht, um welches Message Exchange Pattern es sich bei der Operation handelt.

4.8 Data Contract

Folgend ist die Ansicht eines einzelnen Data Contracts beschrieben.

4.8.1 Code Beispiel [7]

```
data type AddressRecord (
    "street":V<string>,
    "zipCode":V<int>,
    "city":V<string>)

data type MoveHistory
    {"from":AddressRecord, "to":AddressRecord, "when":V<"Date">}

data type CustomerWithAddressAndMoveHistory {
    <<Entity>>"CustomerCoreData":V,
    "AddressRecords":AddressRecord+,
    "MoveHistory": MoveHistory*
}
```

4.8.2 Visualisierung

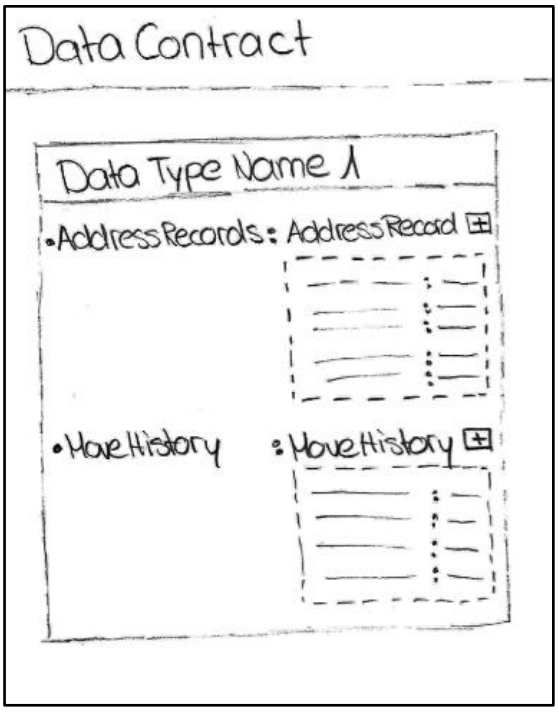

Abbildung 15: Visualisierung der Data Contracts

4.8.3 Beschreibung

Die Ansicht stellt einen Data Contract einer MDSL-Spezifikation textuell dar und ist als Card dargestellt. [19] Falls ein Attribut eines Data Contracts kein Basistyp, sondern ein weiterer Data Contract ist, ist dieser aufklappbar. Zudem werden die Kardinalitäten mit dem dafür vorgesehenen Symbol dargestellt, damit diese auf den ersten Blick erkennbar sind. Diese Symbole werden in einer Legende beschrieben. Weiter wird beim Typ eines Data Contract-Attributes die Elementstruktur anhand des MAP-Icons angegeben. [2] Wie in den Anforderungen definiert, wird nur eine Ansicht bis zur Baumtiefe fünf garantiert, bei der es noch nachvollziehbar ist, wie die Data Contracts verschachtelt sind.

4.8.4 Abhängigkeiten

Bis auf andere Data Contracts, hat ein Data Contract keine Abhängigkeiten.

4.8.5 Begründung

Die Begründung für die Visualisierung mittels Card ist die Gleiche, wie bei den API Providern und Clients. Zusätzlich können Details zu abhängigen Data Contracts direkt in der Card aufgeklappt werden. Somit kann der Benutzer einfach nachvollziehen, welche Contracts miteinander verlinkt sind.

4.9 Gateway

Folgend ist die Ansicht eines API Gateways beschrieben

4.9.1 Code Beispiel [8]

```
API gateway SampleAPIGatweway
  exposes CustomerManagementContract
  at endpoint location "ExternalURI"
  via protocol SOAP_HTTP

  consumes CustomerManagementContract2
  from SampleAPIProvider1
  via protocol gRPC
```

4.9.2 Visualisierung

Abbildung 16: Visualisierung des API Gateways

4.9.3 Beschreibung

Ein API Gateway wird wie praktisch alle anderen Ansichten ebenfalls mit einer Card dargestellt. Die Spezifikation eines API Gateways ist dann in der Card textuell dargestellt.

4.9.4 Abhängigkeiten

Der API Gateway kann Verlinkungen auf Service Endpoints und API Providers enthalten, mit denen zu den entsprechenden Seiten navigiert werden kann.

4.9.5 Begründung

Für die Darstellung der API Gateways wird ebenfalls die Card von Bootstrap verwendet. Der Grund für diesen Entscheid, ist derselbe, wie bei den anderen Seiten, die die Kartenansicht applizieren.

5 Umsetzung

Dieses Kapitel umfasst die Framework- und Toolentscheidungen, sowie die Implementation des «MDSL-Visualizer» und wie das Tool getestet wurde.

5.1 Library- und Toolentscheidungen

Im Projekt mussten wir diverse Entscheidungen treffen, was das Framework und die npm-Module betrifft. Anschliessend sind alle Entscheide dokumentiert. [12]

5.1.1 React [20]

Bei der Auswahl des Frameworks oder der Library kamen nur Angular und React in Frage, weil wir zum einen erste Erfahrungen in diesen sammeln konnten und diese die ausreichende Funktionalität für unser Projekt bieten. Schlussendlich haben wir uns für React entschieden. React wird von Facebook entwickelt und wird bei namhaften Unternehmen eingesetzt. Zum Beispiel findet sie Verwendung bei AirBnb, Netflix und Dropbox. Der Komponentenansatz in React erlaubt es komplexe Probleme in einfache Komponenten aufzuteilen. Dies führt unteranderem zu einer besseren Wiederverwendbarkeit als auch einer besseren Erweiterbarkeit. Diese zwei Faktoren sind Anforderungen im Projekt, die somit abgedeckt werden. Zudem hat React eine grosse Community und man findet bei Problemen schnell und gute Hilfestellungen.

5.1.2 React-Bootstrap [21]

React-Bootstrap ist ein beliebtes und weit verbreitetes Frontend Framework. Für React-Bootstrap haben wir uns entschieden, weil Bootstrap in regelmässigen Abständen aktualisiert und in den gängigen Browser wie Mozilla Firefox oder Google Chrome unterstützt wird. Zusätzlich beinhaltet Bootstrap bereits Komponenten und Styles, die responsiv sind. Somit ersparen wir uns in der Gestaltung der Mobile-Ansicht viel Arbeit.

5.1.3 ANTLR4 [22]

Um eine MDSL-Spezifikation zu parsen, verwenden wir ANTLR4. Diese Entscheidung fiel, weil ANTLR auf diversen Programmiersprachen verfügbar ist und somit die MDSL-Grammatik direkt verwendet werden kann. Zudem wurde es bereits in einem IFS-Projekt erfolgreich eingesetzt.

5.1.4 React Redux [23]

Redux wird verwendet, um einen Store für die ganze Applikation zu haben. Somit kann von jeder Komponente auf diesen Store zugegriffen werden und die Daten müssen nicht explizit an eine Komponente übergeben werden. Für Redux haben wir uns entschieden, weil es die bekannteste Lösung für das Statemanagement in React ist. Ausserdem erspart uns Redux viel unnötigen Code, weil wir von jeder Komponente auf den Store zugreifen können.

5.1.5 TypeScript [24]

Normalerweise gibt es in Javascript keine expliziten Typen, sondern es kann jeder Variable jeder Typ zugewiesen werden. Um uns das Programmieren zu erleichtern, entschieden wir uns, ein Tool für die Typisierung in Javascript einzusetzen. Damit erscheint in der Entwicklungsumgebung ein Fehler bei einer Typverletzung. In der Entscheidungsphase testeten wir Flow und Typescript. Beide Tools unterstützen die Typisierung und können zusammen mit React eingesetzt werden. Die Tools unterscheiden sich grundsätzlich nicht gross. Schlussendlich entschieden wir uns für Typescript, da bei Flow in jeder Datei angegeben werden muss, dass Flow benutzt werden soll. Bei Typescript wird dies über die Dateiendung gelöst.

5.1.6 react-d3-tree [25]

Die Service Endpoints wollen wir in einem Baum visualisieren, dazu haben wir uns zwei verschiedene Tools angeschaut. Neben «react-d3-tree» haben wir auch «react-tree-graph» angeschaut und getestet. Schlussendlich haben wir uns für «react-d3-tree» entschieden, da bei diesem Tool der Baum auch wieder geschlossen und gut gestylt werden kann.

5.1.7 ESLint [26]

ESLint ist ein statisches Code-Analyse Tool, dass wir verwenden, um unsere Code-Qualität zu verbessern. ESLint bietet Support für ES6, JSX und TypeScript an, somit ist ESLint ideal für den Einsatz mit React geeignet. Des Weiteren bietet ESLint die Möglichkeit vereinzelte Regeln zu aktivieren oder eigene Regeln zu implementieren.

5.1.8 Jest [27]

Als Testframework haben wir Jest gewählt. Jest setzen wir ein, weil sie eine gute Unterstützung für React bietet. Dies basiert auf der Tatsache, dass React und Jest von Facebook entwickelt werden. Ausserdem ist Jest sehr mächtig und trotzdem einfach zu schreiben.

5.1.9 react-minimal-pie-chart [28]

Zum Visualisieren der Metriken haben wir ein Kuchendiagramm eingesetzt. Während der Entscheidung haben wir diverse Module getestet. Schlussendlich haben wir uns für dieses Modul entschieden, weil es Typescript unterstützt, was bei den anderen nicht der Fall war. Des Weiteren nimmt es wenig Speicher in Anspruch und verursacht somit keinen grossen Overhead.

5.1.10 react-table [29]

Um die Data Contracts sinnvoll darzustellen und nach diesen zu filtern, haben wir uns entschieden ein Modul einzusetzen. Da die React-Bootstrap Tabelle nur die Anzeige unterstützt, machten wir uns auf die Suche nach einem anderen Tabellen-Modul. Dabei haben wir uns für «react-table» entschieden. Diese bietet eine grosse Anzahl an Funktionen und wird ständig aktualisiert. Dazu verwendeten wir das Package «match-sorter», damit in der Tabelle gefiltert und sortiert werden kann.

5.1.11 Lizenzen

Bei der Auswahl der Tools wurde unteranderem auch auf die Lizenzen geachtet. Als Vorgabe sollten nur Lizenzen von Eclipse, Apache oder MIT verwendet werden. [30] [31] [32] Folgend ist eine Auflistung der eingesetzten Tools mit den Lizenzen zu sehen.

Tool	Lizenz
React	MIT
Bootstrap	MIT
ANTLR4	BSD-3-Clause
Redux	MIT
Typescript	Apache-2.0
react-d3-graph	MIT
ESLint	MIT
Jest	MIT
react-minimal-pie-chart	MIT
react-table	MIT
match-sorter	MIT

Tabelle 3: Tools mit Lizenzen

Bis auf den ANTLR4 haben wir überall eine der drei vorgegebenen Lizenzen verwendet. Da es keinen anderen ANTLR mit einer besseren Lizenz gegeben hat, haben wir nach Rücksprache mit dem Betreuer entschieden, dass trotz der BSD-3-Clause-Lizenz der ANTLR4 verwendet werden darf. [33]

5.2 Vorgehen

In diesem Kapitel wird beschrieben, welchem Vorgehensmodell wir gefolgt sind, wie das zeitliche Vorgehen ist und wie wir vorgegangen sind, um die Risiken zu minimieren.

5.2.1 Vorgehensmodell

In der Studienarbeit sind wir nach einem gemischten Vorgehensmodell namens Scrum++ vorgegangen. Dies ist eine Mischung zwischen Scrum und Unified Process. Durch die Elaborationphase in Unified Process definierten wir in dieser alle Anforderungen, die Architektur und entwarfen ein UI-Konzept. Um trotzdem noch iterativ zu bleiben, kam Scrum während der Projektdauer zum Einsatz.

5.2.2 Zeitliches Vorgehen

Für die Studienarbeit standen für jeden 240 Stunden zur Verfügung. Dies ergibt rund 17 Stunden pro Woche und Person. Die Iterationen setzten wir auf zwei Wochen an und auf Ende jeder Iteration wurde ein Meilenstein definiert. Die detaillierten Informationen sind im Projektplan zu sehen.

5.2.3 Risikominimierung

Zu Beginn der Umsetzung wollten wir vor allem die Risiken minimieren. So entschlossen wir uns, das Parsen einer MDSL-Spezifikation direkt am Anfang zu implementieren. Somit stellten wir sicher, dass wir ohne einen Server auskommen und später nicht noch grosse Überraschungen auf uns zu kommen.

5.3 Implementation

In den nachfolgenden Unterkapitel ist die Implementation der Sprachelemente beschrieben. Zudem ist zu sehen, wie wir das UI-Konzept umgesetzt haben.

5.3.1 ANTLR

Um die hochgeladene MDSL-Spezifikation zu parsen, haben wir ANTLR4 für Javascript eingesetzt. [34] Mit Hilfe des ANTLR4 Tools konnten wir den Lexer, den Parser und einen Tree-Visitor generieren lassen. [13] Um schlussendlich den Parse Tree zu traversieren und die nötigen Informationen zu erhalten, haben wir einen eigenen Visitor implementiert. [14] Der Visitor startet am Beginn der Spezifikation und ruft dann nach und nach alle Kinder auf.

Für die Grafik der Operationen sowie für die Metriken mussten wir die MDSL-Spezifikation anders traversieren. Somit implementierten wir für diese beiden Features auch einen eigenen Visitor, der nicht so grosse Funktionalität besitzt.

```
visitNode(ctx: {
    start: { start: number }; stop: { stop: number }; children: {
        forEach: (arg0: (child: any) => void) => void;
    };
}) {
    if (!ctx) {
        return;
    }

    let tree: MdslTree = {};
    this.mergeContextIntoTree(ctx, tree);
    tree.fileText = this.file.substring(ctx.start.start, ctx.stop.stop + 1);
    tree.strings = [];

    let i: number = 0;
    if (ctx.children) {
        ctx.children.forEach( arg0: (child) => {
            if (child.children && child.children.length !== 0) {
                let ruleName = this.ruleNames[child.ruleIndex];
                (tree[ruleName] || (tree[ruleName] = [])).push(this.visitNode(child));
            } else {
                (tree.strings[i] || (tree.strings[i] = [])).push(child.getText());
                i++;
            }
        });
    }
    return tree;
}
```

Abbildung 18: Implementation des Visitors


```
{…}
endpointContract: (1) […]
  0: {…}
    fileText: "endpoint type HelloWorld"
    name: "HelloWorld"
    start: Object { type: 16, channel: 0, start: 31, … }
    stop: Object { type: 109, channel: 0, start: 45, … }
    strings: Array [ (3) […] ]
    <prototype>: Object { … }
  length: 1
  <prototype>: Array []
fileText: "API description HelloWorldAPI\r\nendpoint type HelloWorld"
name: "HelloWorldAPI"
start: Object { type: 1, channel: 0, start: 0, … }
stop: Object { type: 109, channel: 0, start: 45, … }
strings: Array [ (3) […], (1) […] ]
<prototype>: Object { … }
```


Abbildung 17: Ansicht des generierten Parse Trees in JSON-Format

5.3.2 Menubar

In der Menubar haben wir diverse Navigationselemente. Der Home-Link führt zur Hauptseite. Der Help-Link zeigt in einem Popup eine Hilfe an. Der dritte Link ist je nach Seite unterschiedlich und verweist jeweils auf die entsprechende MDSL-Webseite. Zu guter Letzt haben wir den Download-Link, womit die geänderte MDSL-Spezifikation heruntergeladen werden kann.

Abbildung 19: Screenshot der Menubar

5.3.3 Navigation

Die Navigation haben wir als Breadcrumb umgesetzt. Die aktuelle Seite wird so stets zuvorderst in die Navigation übernommen. Bei der Anzeige gibt es Links auf die vorherigen Seiten, auf die direkt navigiert werden kann.

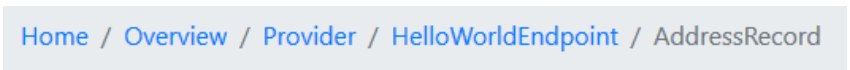

Abbildung 20: Screenshot der Breadcrumb-Navigation

5.3.4 Hauptseite

Bei der Hauptseite hat der User die Möglichkeit seine zu analysierende MDSL-Spezifikation auszuwählen. [1] Anschliessend kann er auf den Upload-Button klicken. Augenblicklich wird eine Statistik ausgegeben. Die Anzahl der verschiedenen genutzten Sprachelemente werden in einem Kreisdiagramm visualisiert. Durch dieses Visualisierungselement sieht der Nutzer auf einem Blick, wie die Sprachelemente im Verhältnis stehen. Sollte ein vom Tool unterstützendes Sprachelement in der Spezifikation nicht vorkommen, so wird das Element in der Grafik gar nicht erst angezeigt. Unterhalb des Kreises sind noch zusätzliche Informationen ersichtlich, wie die maximale unterstützende Data Contracts Tiefe. Die farblich unterscheidenden Elemente sind klickbar und verlinken den Benutzer direkt zu der jeweiligen Ansicht. Möchte der Nutzer zuerst zu der Übersicht navigieren, so klickt er auf den Button «Display API Description».

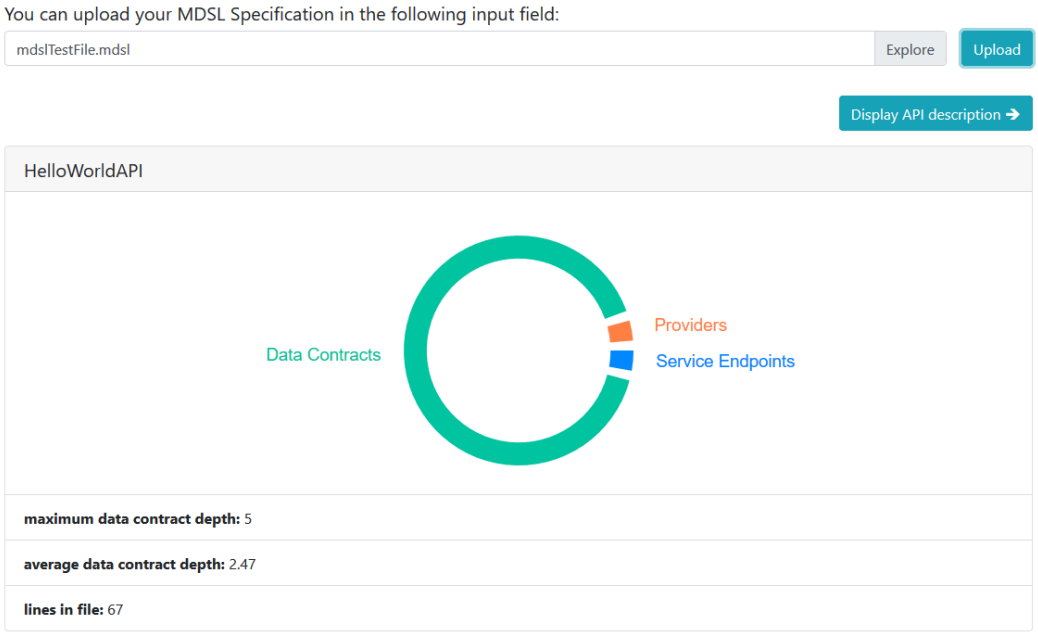

Abbildung 21: Screenshot der Main Page

Sollte die zu visualisierende MDSL-Spezifikation fehlerhaft sein, so wird dem User eine Liste aller Fehler ausgegeben.

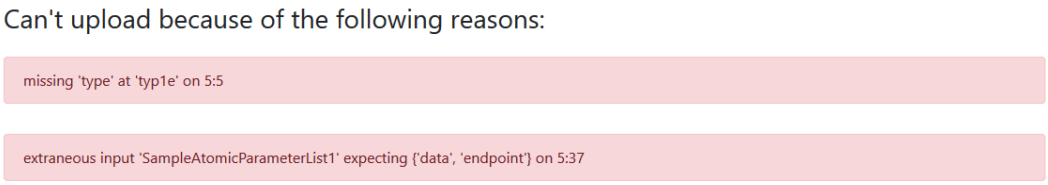

Abbildung 22: Screenshot der Main Page mit Errors

5.3.5 Overview

Die Overview haben wir getreu den UI-Skizzen umgesetzt. Oben auf dem Screenshot ist die API Description zu sehen mit Links zu den Microservice API Patterns und sowie die dazugehörigen Icons. [2] In der Mitte ist die «hexagoniong»-Grafik mit dem Client und dem Provider zu sehen. [17] Mit einem Klick auf das Hexagon kommt man jeweils zu der Einzelansicht. Unten gibt es eine Übersicht über alle definierten Data Contracts. In der Tabelle kann man auf die nächsten Seiten navigieren, sowie nach einem bestimmten Data Contract suchen.

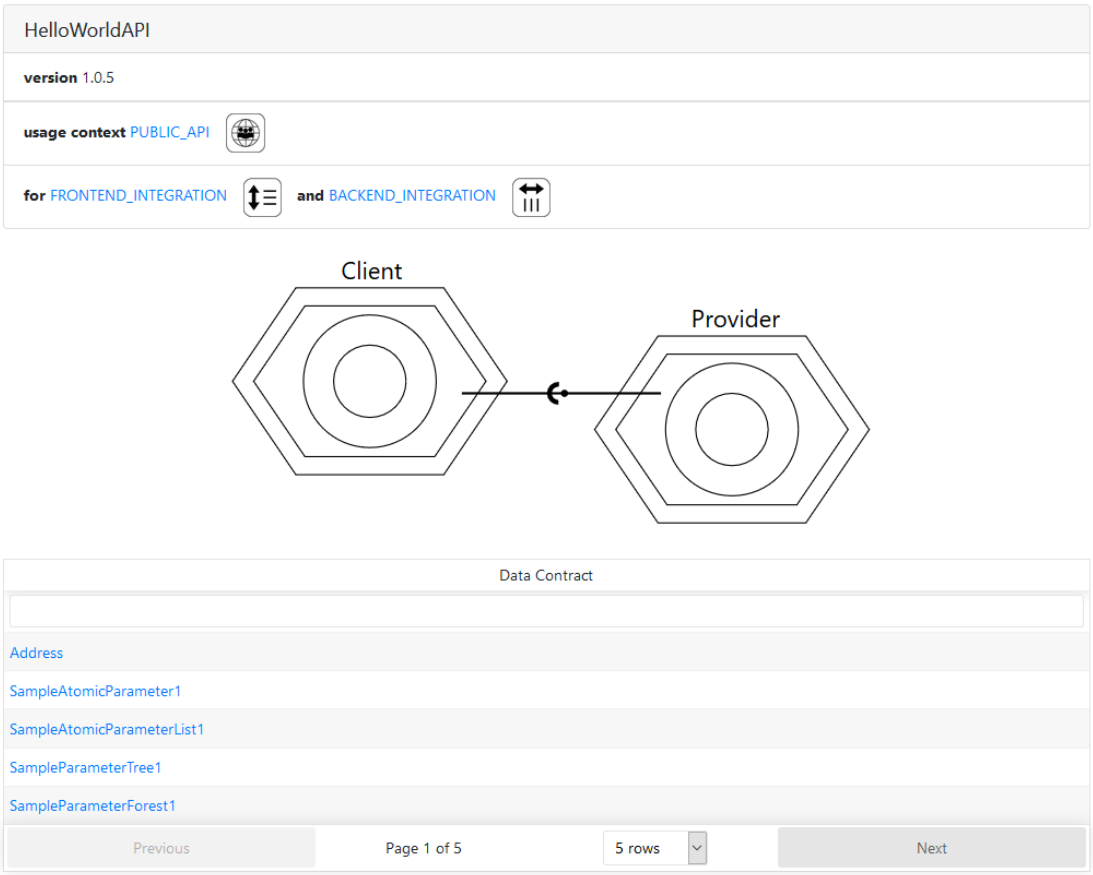

Abbildung 23: Screenshot der Overview

5.3.6 Provider

Beim Provider werden links in der Card die Informationen dargestellt. Der offerierte Service Endpoint ist zudem verlinkt. Auf der rechten Seite ist ein Feature zu finden, welches nicht von Beginn an geplant war. So sieht man dort den Provider in textueller Form, der bearbeitet und gespeichert werden kann. Beim Speichern wird der Provider neu geparst und die Änderungen werden umgehend dargestellt.

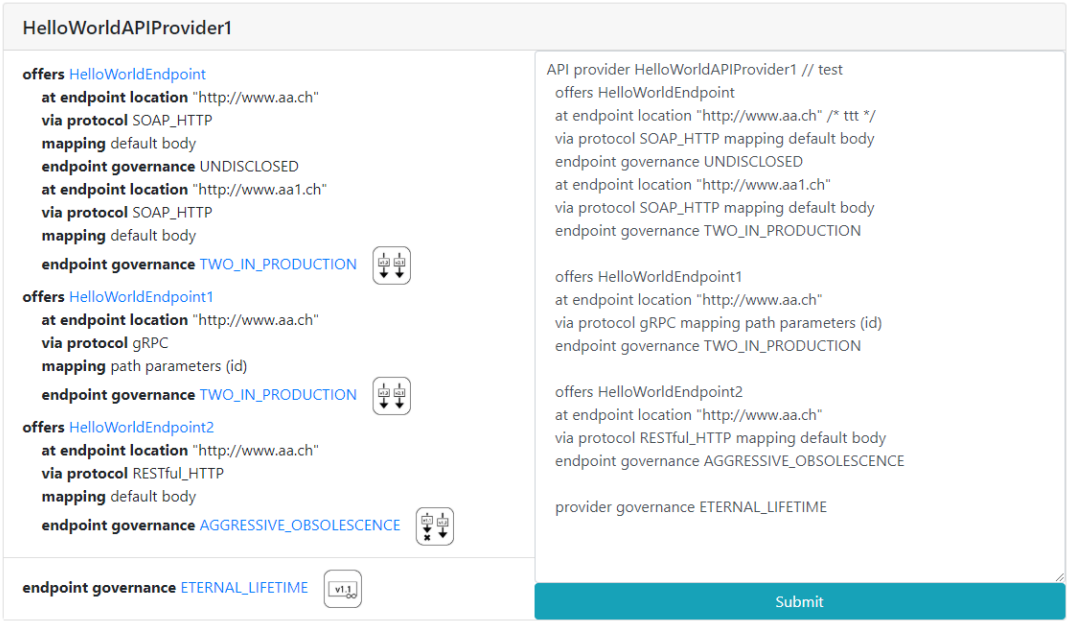

Abbildung 24: Screenshot eines API Providers

Falls kein Provider vorhanden ist, wird der Benutzer mit einer Meldung auf dies aufmerksam. Dies ist bei allen folgenden MDSL-Elementen gleich implementiert.

Abbildung 25: Meldung, wenn kein Provider vorhanden ist

5.3.7 Client

Die einzelnen Eigenschaften eines API Clients wie zum Beispiel der Service Contract oder der Provider sind klickbar. Beim Klick auf einen Link öffnet sich die Ansicht zu dem Element. Ein Client kann auch mehrere Konsumenten haben, die in einem nächsten Listeneintrag in der gleichen Card angezeigt werden. Rechts in der Kartenansicht besteht jeweils die Möglichkeit, die Spezifikation des Clients zu bearbeiten. Die getätigten Änderungen werden beim Speichern zugleich in die Card übernommen.

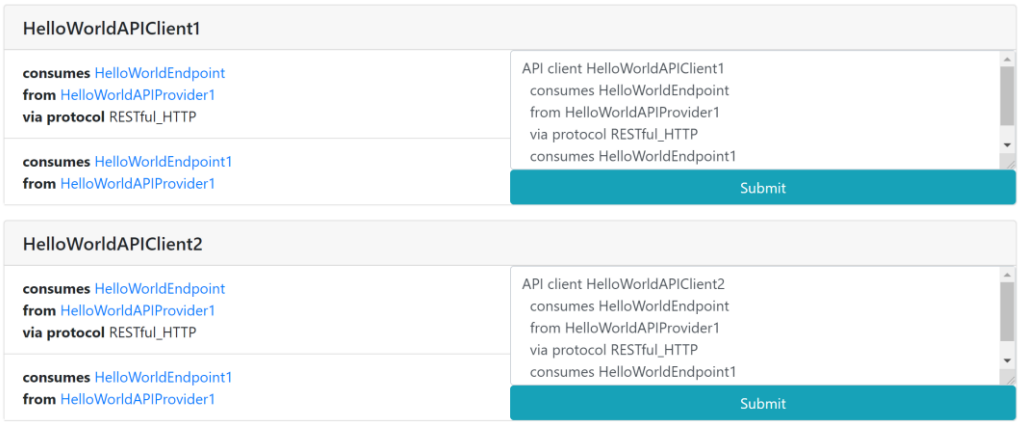

Abbildung 26: Ansicht der API Clients

5.3.8 Service Contract

Der Service Contract war von den Platzverhältnissen am schwierigsten umzusetzen, da dieser viele Informationen beinhaltet. Zuoberst werden die allgemeinen Informationen zum Service Contract dargestellt. Unterhalb sieht man alle Operationen, die der Service Contract anbietet. Diese werden per Klick auf der rechten Seite geöffnet, wo die Details zur Operation zu sehen sind. Falls bei der Operation ein Data Contract erwartet oder geliefert wird, wird er entweder angezeigt oder verlinkt. Ganz oben ist ein Button zum Editieren des Service Endpoints zu finden. Das Editieren findet auf einer anderen Seite statt, weil der Platz auf dieser Seite mangelhaft ist.

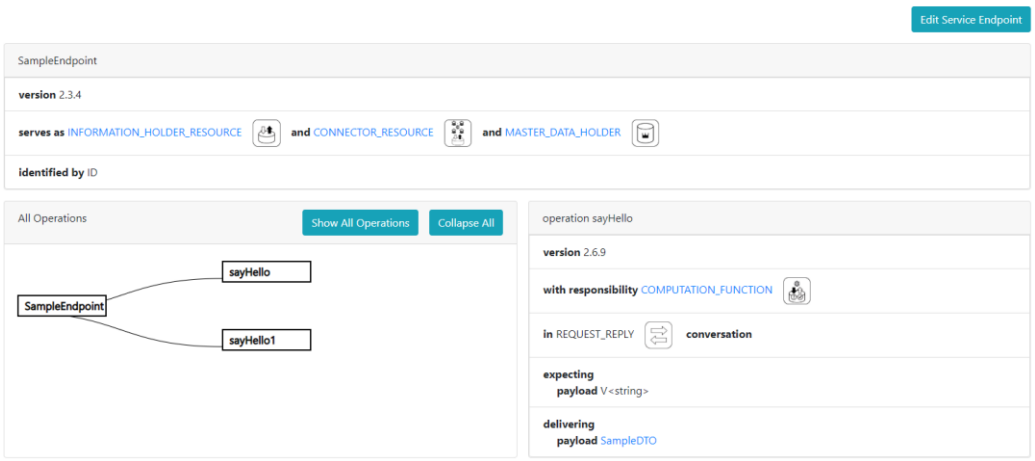

Abbildung 27: Screenshot eines Service Contracts

5.3.9 Data Contract

Zu den Data Contracts gibt es eine Tabelle, in der alle aufgelistet sind. Die Tabelle verfügt über eine Sortierfunktion, die beim Klick auf den Tabellentitel ausgeführt wird. Des Weiteren verfügt die Tabelle über eine Filterfunktion, bei welcher der zu suchende Data Contract im Inputfeld unterhalb des Tabellentitels eingegeben werden kann. Zudem kann die Anzahl der Tabelleneinträge variiert werden.

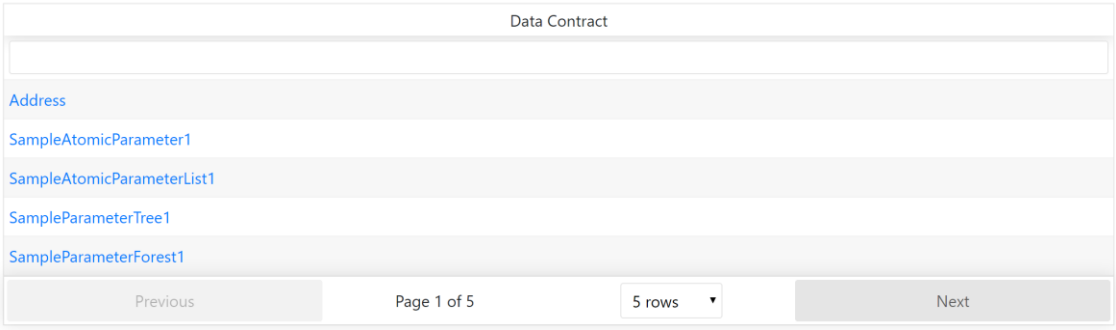

Abbildung 28: Tabelle mit allen Data Contracts

Nach der Auswahl eines Eintrags öffnet sich die Ansicht des Data Contracts. Zu Beginn wird zwischen den verschiedenen Elementstrukturen Single Parameter Node, Atomic Parameter List, Parameter Forest und Parameter Tree unterschieden. Für jede Elementstruktur wurde eine eigene React-Komponente erstellt. Dabei mussten diverse Elemente wie die Version, der Stereotyp oder die Kardinalität beachtet werden, weil diese je nach Elementstruktur verschieden sind.

Die Visualisierung der verschiedenen Elementstrukturen ist immer gleich aufgebaut. Es wird eine Card verwendet. Auf der linken Seite der Kopfzeile wird der Name des Data Contracts angezeigt. Auf der rechten Seite findet der User Informationen, wie beispielsweise die Version. Im Body der Card werden die einzelnen Parameter aufgelistet. Zusätzlich wurde auch hier mit Verlinkungen gearbeitet.

Der Single Parameter Node kann entweder einen generischen Parameter, einen Atomic Parameter oder eine Typreferenz haben. Eine Typreferenz bedeutet, dass auf einen beliebigen Data Contract referenziert wird. Sollte eine Typreferenz vorhanden sein, dann ist diese aufklappbar.

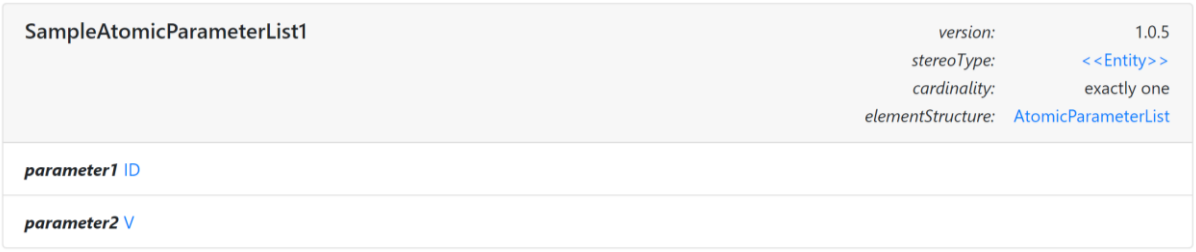

Abbildung 29: Beispiel einer Atomic Parameter List

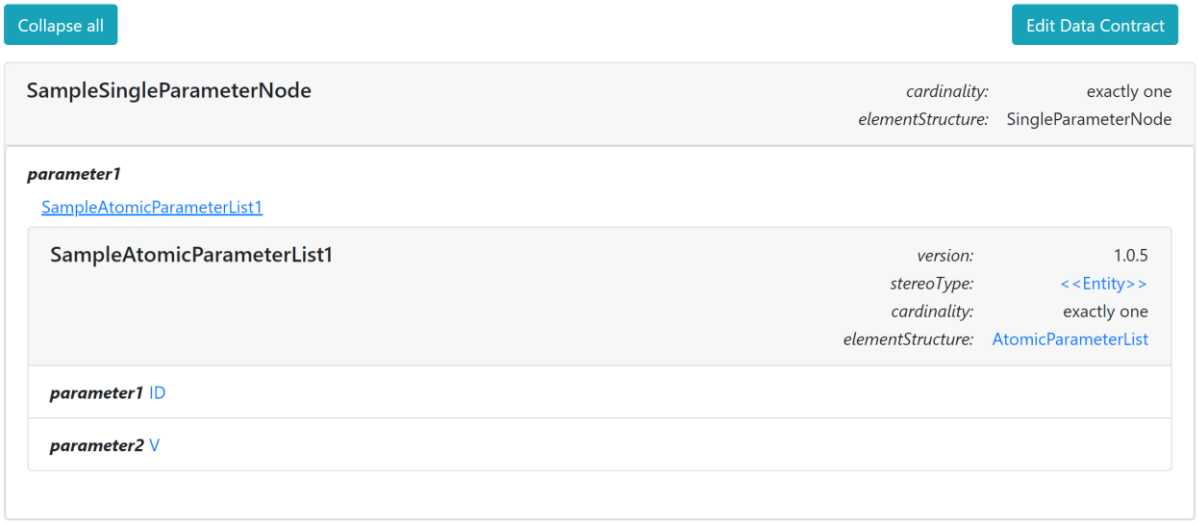

Abbildung 30: Beispiel eines Single Parameter Nodes

Gemäss der obigen Grafik sieht der Nutzer nun die referenzierte Elementstruktur. Klickt der User nochmals auf die Typreferenz, verschwindet diese wieder. Beim Klicken auf den «Collapse all»-Button werden alle aufgeklappten Verschachtelungen wieder ausgeblendet. Der «Edit Data Contract»-Button existiert in jedem Data Contract. Beim Drücken dieses Buttons bekommt der Anwender die Möglichkeit die Spezifikation zu bearbeiten.

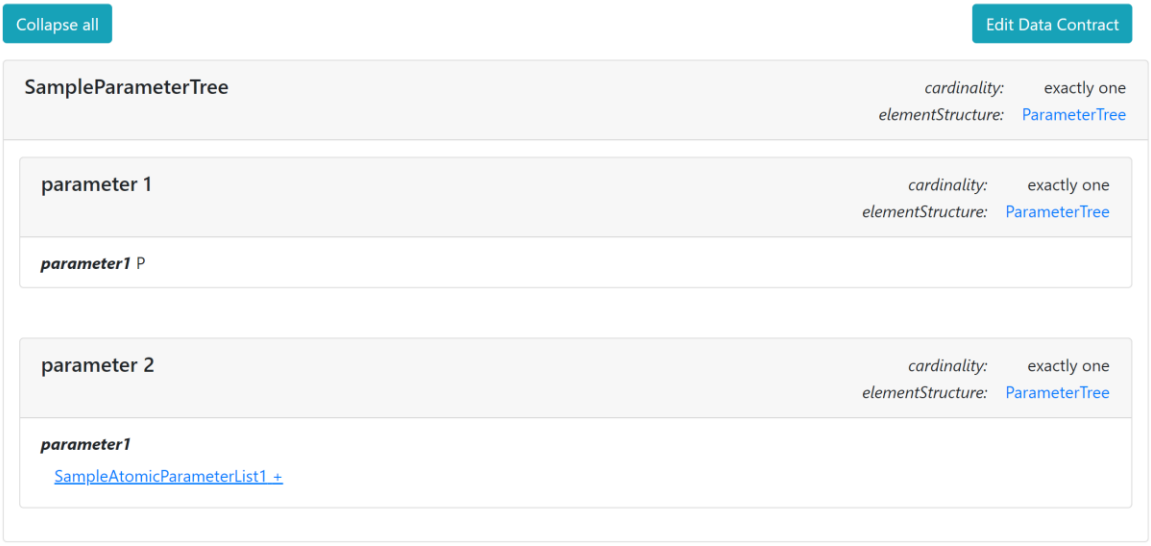

Abbildung 31: Beispiel eines Parameter Trees

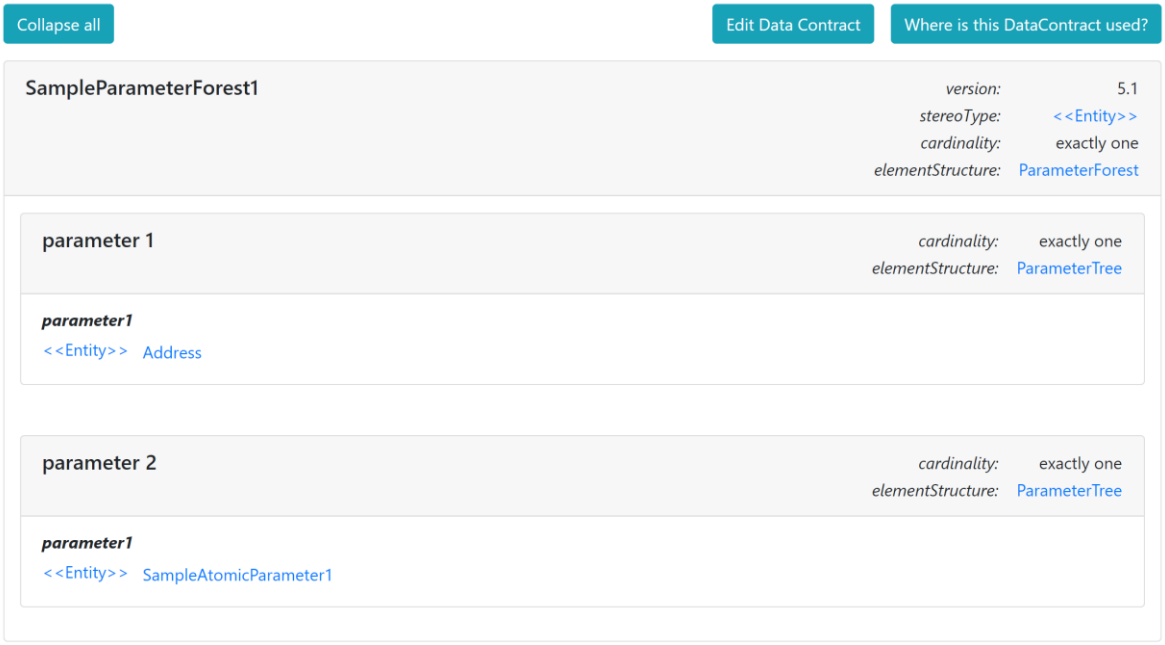

Abbildung 32: Beispiel eines Parameter Forests

Falls der Button mit der Beschriftung «Where is this Data Contract used?» eingeblendet wird, bedeutet das, dass es Operationen gibt, die auf diesen Data Contract zeigen. Bei der Betätigung dieses Buttons werden diese Abhängigkeiten angezeigt. Der aufgelistete Service Endpoint ist ein Link und führt zur Visualisierung dieses Sprachelements.

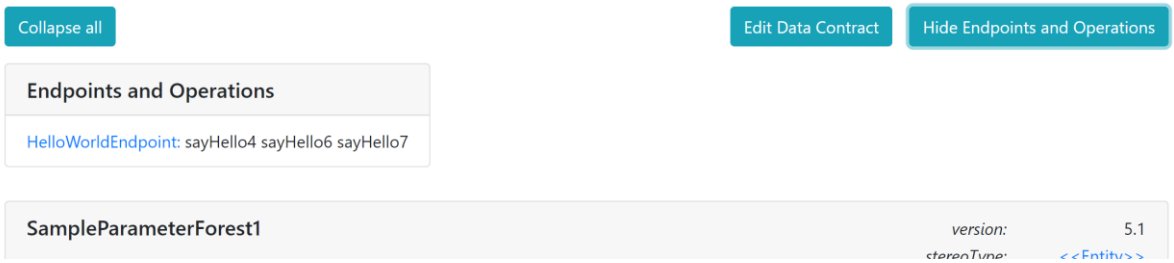

Abbildung 33: Abhängigkeiten zu Operationen

5.4 Testing

Während des Projektes wurden diverse Tests durchgeführt. Anschliessend werden die verschiedenen Tests genauer beschrieben.

5.4.1 Unit Test

Auf das Schreiben von Unit Tests haben wir grösstenteils verzichtet, weil bei einer solchen User-Interface lastigen Applikation ist es sehr schwierig gute Unit Tests zu schreiben. Es bestände die Möglichkeit jede Komponente in einem Unit Test zu rendern, allerdings würde dieser Test nichts aussagen. Die Aussage wäre lediglich, dass die Komponente ohne einen Fehler aufgerufen werden kann. Einzig für die Business-Logik haben wir Unit-Tests eingesetzt, da dies nicht direkt das User-Interface betrifft.

5.4.2 Usability Test

Um die Bedienbarkeit und die Gebrauchstauglichkeit zu testen, haben wir zwei Usability Tests durchgeführt. Den ersten Usability Test haben wir in der Mitte der Entwicklungsphase angesetzt. Der Gesamteindruck war gut, allerdings konnten wir viele Verbesserungs- und Erweiterungsmöglichkeiten aus dem Usability Test für unserer Applikation ziehen. Nachdem wir viele dieser Möglichkeiten umgesetzt haben, führten wir gegen Ende der Konstruktion den zweiten Usability Test durch. Somit konnten wir noch eine weitere Meinung einholen, durch die wir unsere Applikation verbessern konnten. Die vollständig dokumentierten Usability Tests sind in Anhang 11.4 zu finden.

5.4.3 System Test

Während des Projektes wurden stetig System Tests durchgeführt. So wurde jeweils während dem Programmieren der einzelnen Komponenten die Funktionalität ausgiebig mit selbst erstellten MDSL-Spezifikationen getestet. Vor einem Commit sind nochmals alle veränderten Komponenten auf Fehlverhalten geprüft worden. Als die Programmierarbeiten gestoppt wurden, haben wir beide die volle Funktionalität nochmals geprüft und bewusst auf Fehlverhalten der Applikation geprüft. Eine Beispiel-Spezifikation ist auf der nächsten Seite abgebildet.

```
API description ProcuratioAPI
    version 1.0.5
    usage context PUBLIC_API for BACKEND_INTEGRATION

data type Client {
    "name": V<string>,
    "street": V<string>,
    "houseNumber": V<int>,
    "postalCode": V<int>,
    "city": V<string>
}
data type Item {
    "name": V<string>
}
data type Order {
    "client": Client,
    "item": Item
}

endpoint type ClientEndpoint
exposes
  operation createClient
    expecting payload Client
    delivering payload {"statusCode": V<int>}
  operation updateClient
    expecting payload Client
    delivering payload {"statusCode": V<int>}
endpoint type ItemEndpoint
exposes
  operation createItem
    expecting payload Item
    delivering payload {"statusCode": V<int>}
  operation updateItem
    expecting payload Item
    delivering payload {"statusCode": V<int>}
endpoint type OrderEndpoint
exposes
  operation createOrder
    expecting payload Order
    delivering payload {"statusCode": V<int>}
  operation updateOrder
    expecting payload Order
    delivering payload {"statusCode": V<int>}

API provider ProcuratioAPIProvider
  offers ClientEndpoint
    at endpoint location "http://www.procuratio.ch:40000/clients"
    via protocol RESTful_HTTP
  offers ItemEndpoint
    at endpoint location "http://www.procuratio.ch:40000/items"
    via protocol RESTful_HTTP
  offers OrderEndpoint
    at endpoint location "http://www.procuratio.ch:40000/orders"
    via protocol RESTful_HTTP
provider governance ETERNAL_LIFETIME

API client ProcuratioAPIClient
  consumes ClientEndpoint
  from ProcuratioAPIProvider
  via protocol RESTful_HTTP

  consumes ItemEndpoint
  from ProcuratioAPIProvider
  via protocol RESTful_HTTP

  consumes OrderEndpoint
  from ProcuratioAPIProvider
  via protocol RESTful_HTTP
```

Abbildung 34: Beispiel einer MDSL-Spezifikation

6 Reflexion

Im Kapitel Reflexion bieten wir eine Übersicht über die Ergebnisse, eine Anforderungsanalyse, einen Vergleich mit OpenAPI Specification, eine kurze Zusammenfassung sowie einen Ausblick in Erweiterungsmöglichkeiten.

6.1 Ergebnisse

Im Rahmen dieser Studienarbeit ist ein Visualisierungstool entstanden, welches den Namen «MDSL Visualizer» trägt. Das Tool ermöglicht es seinem Nutzer eine MDSL-Spezifikation darzustellen. Nachdem die MDSL-Spezifikation hochgeladen wurde, kann die Datei durch einen Klick auf den Upload-Button geparst werden. Im unteren Bereich der Hauptseite wird eine Metrik ausgegeben, die Eckdaten zu der Spezifikation liefert. Der User kann nun über einen Button zur Übersicht navigieren. Er hat ebenfalls die Möglichkeit die Ansichten der einzelnen Sprachelementen in einer neuen Seite zu öffnen. In der Übersicht findet der Anwender die API Description, eine hexagonale Grafik, und die Tabelle mit allen Data Contracts. Die Grafik besteht aus zwei Hexagons. Eines führt zu der Ansicht der API Clients und das andere zu den API Providern. Für die Visualisierung der Clients und den Providern wurde auf die Kartenansicht gesetzt. Alle Details werden in der Karte festgehalten. Bei der Darstellung der Data Contracts wurde zusätzlich zu der Kartenansicht, mit einem Einblendmechanismus gearbeitet. Weil die Beziehung zwischen Endpoints und Operationen einer Baumstruktur ähnelt, haben wir uns dort für eine Baumvisualisierung entschieden. Die einzelnen Blätter stellen dabei die Operationen dar. Die Details können durch einen Klick eingeblendet werden.

6.2 Anforderungsanalyse

Im diesem Unterkapitel werden die funktionalen und nicht-funktionalen Anforderungen analysiert, ob diese umgesetzt wurden. Zudem beinhaltet es eine Evaluation, welche Teile der MDSL-Grammatik umgesetzt wurden.

6.2.1 Funktionale Anforderungen

Bei den funktionalen Anforderungen haben wir die User Story umgesetzt, in der wir ein Visualisierungstool für MDSL-Spezifikationen implementiert haben. Zu der Unterstützung von CML-Spezifikationen kamen wir leider nicht mehr, weil wir somit die Möglichkeit hatten noch mehr Funktionen für das eigentliche Tool anzubieten.

6.2.2 Nicht-funktionale Anforderungen

Folgend werden die nicht-funktionalen Anforderungen analysiert, die durch uns getestet werden konnten. Die Anforderungen in den Bereichen Suitability und Maintainability konnten wir nicht selbst testen, da dies fremden Benutzerinput benötigt.

Performance

Nach dem Hochladen hat es bei uns stets weniger als eine Sekunde gedauert, bis die MDSL-Spezifikation visuell dargestellt wurde. Somit haben wir die Anforderung erfüllt.

Accuracy

Es werden nur MDSL-Spezifikationen dargestellt, bei denen es beim Parsen keinen Fehler gegeben hat. Zudem gingen während unseren Usability-Tests keine Informationen verloren und es wurde stets alles angezeigt.

Reliability

Während der Usability-Tests gab es keine Abstürze der Applikation. Dies wurde anhand der vorgegebenen MDSL-Spezifikation sowie durch eigene MDSL-Spezifikationen getestet.

Usability

Es wurde ein Usability Test mit unserem Betreuer und einem weiteren Mitarbeiter des Instituts für Software durchgeführt und die gewünschten Änderungen wurden umgesetzt.

Transferability

Wenn der Installationsanleitung gefolgt wird, ist es möglich, die Applikation in maximal fünf Minuten zum Laufen zu bringen.

Portability

Es wurde darauf geachtet, dass alle Ansichten auch responsive sind. Somit sollte es kein Problem sein, MDSL-Spezifikationen auf dem Smartphone oder Tablet anzusehen.

Compatibility (optional)

Dieses optionale Feature konnten wir leider nicht umsetzen, weil wir uns für eine Architektur ohne Persistenz entschieden haben. Ohne die Persistenz lässt sich keine MDSL-Spezifikation speichern. Dadurch kann nicht auf eine Spezifikation zugegriffen werden, solange diese nicht über die Applikation hochgeladen wurde.

6.2.3 Evaluation

In den zwei nachfolgenden Unterkapitel wird der Status der Aufgaben festgehalten.

6.2.3.1 Hohe Priorität

In dieser Tabelle sind die Aufgaben aufgelistet, die mit der höchsten Priorität versehen wurden.

Teile	Erfüllt (ja/nein)
MDSL Grammar Quick Reference (Skeletons) [6] Service Contract Skeleton	ja
Service Endpoint Contracts in MDSL [5] Concepts	ja
Data Contracts and Schemas in MDSL [7]	ja
Runtime Language Concepts: API Provider, API Client, API Gateway [8] API Provider	ja
Runtime Language Concepts: API Provider, API Client, API Gateway API Client	ja

Tabelle 4: Sprachelemente mit hoher Priorität

In dieser Einstufung haben wir alle Aufgaben meistern können.

6.2.3.2 Mittlere Priorität

Die unten aufgeführten Aufgaben sind als mittele Priorität eingestuft worden.

Teile	Erfüllt (ja/nein)
MDSL Grammar Quick Reference (Skeletons) Reporting	nein
Service Endpoint Contracts in MDSL Concepts (identified by)	ja*
Service Endpoint Contracts in MDSL Security Policy	nein
Service Endpoint Contracts in MDSL Parameterbinding	ja*
Runtime Language Concepts: API Provider, API Client, API Gateway API Provider - SLA	nein
Runtime Language Concepts: API Provider, API Client, API Gateway API Gateway	nein

Tabelle 5: Sprachelemente mit mittlerer Priorität

* nur textuelle Anzeige

Wir haben nicht viele der mittleren Prioritäten umgesetzt, weil wir die Zeit in ungeplante Features investiert haben, wie zum Beispiel das Editieren der Spezifikation oder die Metriken.

6.3 Vergleich mit OpenAPI Specification (Swagger)

Abschliessend vergleichen wir unser Visualisierungstool für MDSL-Spezifikationen mit Swagger UI. OpenAPI Specification (OAS) ermöglicht es RESTful HTTP API's zu beschreiben. [35] Vorerst erläutern wir kurz die groben Unterschiede zwischen MDSL und OpenAPI Specification und anschliessend die der Visualisierungen.

Während MDSL diverse Integrationstechnologien wie RESTful HTTP, Jolie oder WSDL unterstützt, können mit OpenAPI Specification nur RESTful HTTP-Schnittstellen beschrieben werden. [1] Weiter ist der Aufbau der Sprache unterschiedlich, bei OAS liegt der Schwerpunkt auf den Operationen sprich den URI's, während bei MDSL der Schwerpunkt auf den Ressourcen Endpoints liegt.

Aufgrund des verschiedenen Aufbaus der Sprachen selbst, variiert auch das User-Interface. So stehen bei Swagger UI die URI's mit den jeweiligen HTTP-Methoden im Mittelpunkt. Diese sind auf den ersten Blick ersichtlich und erscheinen in unterschiedlichen Farben. Wohingegen bei unserem Tool vorerst die API Beschreibung zu sehen ist und erst später sind die Operationen unter dem Service Contract ersichtlich. Beim Vergleich der Operationen zwischen den beiden Tools, werden diese bei uns als Baum dargestellt und bei Swagger UI ist es lediglich eine Auflistung. Im Gegensatz zu uns ist es in Swagger UI möglich, direkt einen Request auf die API zu senden und somit zu testen. In der Auflistung der Datenrepräsentationen gibt es keine grossen Unterschiede. Beide Tools stellen diese bereits auf der Startseite dar.

6.4 Zusammenfassung

Zusammenfassend lässt sich sagen, dass wir alle Aufgaben der höchsten Priorität implementieren konnten. Zusätzlich fanden wir noch Zeit, um gewünschte Features zu programmieren. Beispielsweise haben wir die statistische Auswertung umgesetzt, welche nicht von Beginn an geplant war. Da wir eine Single Page Applikation entwickelt haben, hat sich der Usability Test in der Mitte der Construction Phase als sehr hilfreich herausgestellt. Das Feedback zeigte uns auf, in welchen Bereichen wir noch Verbesserungspotenzial haben. Zum Ende der Construction Phase wurde der Code nochmals akribisch durchgeschaut. Der Code wurde anschliessend dementsprechend überarbeitet. Rückblickend lässt sich des Weiteren sagen, dass die Implementierung der Data Contracts sich als mühselig herausstellte. Der Grund dafür waren die vielen verschiedenen Verschachtelungsmöglichkeiten. Schlussendlich sind wir mit unserer geleisteten Arbeit und mit dem Endprodukt zufrieden.

6.5 Ausblick

Unserem Visualisierungstool steht einem produktiven Einsatz grundsätzlich nichts im Wege. Das einzige Problem, das Entstehen könnte, wäre das nicht alle Informationen der MDSL-Spezifikation dargestellt werden. Dies ist der Fall, weil wir nicht alle MDSL-Sprachelemente implementiert haben.

In Zukunft könnte unser Visualisierungstool in verschiedenen Bereichen erweitert werden. So könnten zum Beispiel neue Sprachelemente von MDSL im Tool unterstützt werden. Eine andere Erweiterungsmöglichkeit wäre, das Visualisierungstool mit einem Backend zu versehen. Somit ist es möglich, die MDSL-Spezifikationen zu persistieren. Dadurch besteht die Möglichkeit, eine MDSL-Spezifikation direkt via Link zu öffnen oder eine MDSL-Spezifikation auf einer anderen Webseite einzubinden.

7 Abbildungsverzeichnis

9 Literaturverzeichnis

[1] O. Zimmermann, „MDSL," [Online]. Available: https://socadk.github.io/MDSL/index. [Zugriff am 16. Oktober 2019].

[2] O. Zimmermann, M. Stocker, U. Zdun, D. Lübke und C. Pautasso, „Microservice API Patterns," [Online]. Available: https://microservice-api-patterns.org/. [Zugriff am 16. Oktober 2019].

[3] O. Zimmermann, „Aufgabenstellung Studienarbeit".

[4] „Context Mapper," [Online]. Available: https://contextmapper.org/. [Zugriff am 16. Dezember 2019].

[5] O. Zimmermann, „Service Endpoint Contracts in MDSL," [Online]. Available: https://socadk.github.io/MDSL/servicecontract. [Zugriff am 16. Oktober 2019].

[6] O. Zimmermann, „MDSL Grammar Quick Reference (Skeletons)," [Online]. Available: https://socadk.github.io/MDSL/quickreference. [Zugriff am 16. Dezember 2019].

[7] O. Zimmermann, „Data Contracts and Schemas in MDSL," [Online]. Available: https://socadk.github.io/MDSL/datacontract. [Zugriff am 16. Oktober 2019].

[8] O. Zimmermann, „Runtime Language Concepts: API Provider, API Client, API Gateway," [Online]. Available: https://socadk.github.io/MDSL/optionalparts. [Zugriff am 16. Oktober 2019].

[9] O. Zeigermann und N. Hartmann, „Flux-Architektur am Beispiel von Redux," in *React-Die praktische Einführung in React, React Router und Redux*, Heidelberg, dpunkt.verlag, 2016, p. 342.

[10] „React.Component - React," [Online]. Available: https://reactjs.org/docs/react-component.html. [Zugriff am 23. Oktober 2019].

[11] „ECMAScript 6," [Online]. Available: https://www.w3schools.com/js/js_es6.asp. [Zugriff am 06. November 2019].

[12] „npm | build amazing thins," [Online]. Available: https://www.npmjs.com/. [Zugriff am 16. Dezember 2019].

[13] „ANTLR Tool Command Line Options," [Online]. Available: https://github.com/antlr/antlr4/blob/master/doc/tool-options.md. [Zugriff am 06. Dezember 2019].

[14] E. Gamma, R. Helm, R. Johnson und J. Vlissides, Design patterns : elements of reusable object-oriented software, Addison Wesley, 1995.

[15] „Docker," [Online]. Available: https://www.docker.com/. [Zugriff am 28. Oktober 2019].

[16] herbertograca, „DDD, Hexagonal, Onion, Clean, CQRS,“ [Online]. Available: https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/. [Zugriff am 06. November 2019].

[17] O. Zimmermann, „DOMAIN-SPECIFIC SERVICE DECOMPOSITION WITH MICROSERVICE API PATTERNS,“ [Online]. Available: https://www.conf-micro.services/2019/slides//keynotes/Zimmerman.pdf. [Zugriff am 12. Dezember 2019].

[18] „Breadcrumb - React Bootstrap,“ [Online]. Available: https://react-bootstrap.github.io/components/breadcrumb/. [Zugriff am 16. Oktober 2019].

[19] „Cards - React Bootstrap,“ [Online]. Available: https://react-bootstrap.github.io/components/cards/. [Zugriff am 16. Oktober 2019].

[20] „react - npm,“ [Online]. Available: https://www.npmjs.com/package/react. [Zugriff am 23. Oktober 2019].

[21] „react-bootstrap - npm,“ [Online]. Available: https://www.npmjs.com/package/react-bootstrap. [Zugriff am 23. Oktober 2019].

[22] «antlr4 - npm,» [Online]. Available: https://www.npmjs.com/package/antlr4. [Zugriff am 23. Oktober 2019].

[23] „react-redux - npm,“ [Online]. Available: https://www.npmjs.com/package/react-redux. [Zugriff am 23. Oktober 2019].

[24] „typescript - npm,“ [Online]. Available: https://www.npmjs.com/package/typescript. [Zugriff am 23. Oktober 2019].

[25] „react-d3-tree - npm,“ [Online]. Available: https://www.npmjs.com/package/react-d3-tree. [Zugriff am 23. Oktober 2019].

[26] „eslint - npm,“ [Online]. Available: https://www.npmjs.com/package/eslint. [Zugriff am 23. Oktober 2019].

[27] „jest - npm,“ [Online]. Available: https://www.npmjs.com/package/jest. [Zugriff am 23. Oktober 2019].

[28] „react-minimal-pie-chart - npm,“ [Online]. Available: https://www.npmjs.com/package/react-minimal-pie-chart. [Zugriff am 02 Dezember 2019].

[29] „react-table - npm,“ [Online]. Available: https://www.npmjs.com/package/react-table. [Zugriff am 02 Dezember 2019].

[30] „Eclipse Public License,“ [Online]. Available: https://opensource.org/licenses/EPL-2.0. [Zugriff am 28. Oktober 2019].

[31] „Apache License,“ [Online]. Available: https://opensource.org/licenses/Apache-2.0. [Zugriff am 28. Oktober 2019].

[32] „The MIT License,“ [Online]. Available: https://opensource.org/licenses/MIT. [Zugriff am 28. Oktober 2019].

[33] „The 3-Clause BSD License,“ [Online]. Available: https://opensource.org/licenses/BSD-3-Clause. [Zugriff am 28. Oktober 2019].

[34] „Javascript antlr4,“ [Online]. Available: https://github.com/antlr/antlr4/blob/master/doc/javascript-target.md. [Zugriff am 06. Dezember 2019].

[35] „Swagger,“ [Online]. Available: https://swagger.io/. [Zugriff am 16. Dezember 2019].

10 Tabellenverzeichnis

11 Anhänge

11.1 Installationsanleitung

Die Applikation kann entweder über npm oder via Docker installiert und gestartet werden.

11.1.1 npm

Um die Applikation zu installieren muss im Projektverzeichnis zuerst folgender Befehl ausgeführt werden:

```
npm install
```

Wenn die Installation fertig ist, kann die Applikation mit folgendem Befehl gestartet werden:

```
npm start
```

Nun sollte die Applikation unter http://localhost:3000 erreichbar sein.

11.1.2 Docker

Das Dockerfile liegt im Hauptverzeichnis des Projektes. Zuerst muss der Docker-Containter gebuildet werden. Dazu in der Konsole ins Projektverzeichnis navigieren und folgenden Befehl ausführen.

```
docker build -t mdsl-visualizer
```

Anschliessend kann der Docker-Container mit folgendem Befehl gestartet werden.

```
docker run -it --rm -p 80:3000 mdsl-visualizer
```

Nun sollte die Applikation unter http://localhost erreichbar sein.

11.1.3 Aktualisierung ANTLR4-Datei

Falls die ANTLR4-Datei aktualisiert wird, können die Dateien für das Parsing mit Hilfe des ANTLR4 Tools generiert werden. Das ANTLR4 Tool kann auf der Webseite heruntergeladen werden (https://www.antlr.org/download.html). Anschliessend folgender Befehl in der Konsole ausführen.

```
antlr4 -Dlanguage=JavaScript MyGrammar.g4 -no-listener -visitor
```

Nun können die generierten Dateien im Projektverzeichnis src/antlr4/generatedAntlr4Files ersetzt werden.