

Inhaltsverzeichnis

1	Konzeption des Prototyps	2
1.1	Datenmodellierung	2
1.2	Architektur	5
1.3	Vorgehensmodell	5

1 Konzeption des Prototyps

1.1 Datenmodellierung

Die zentrale Datenstruktur der Applikation ist die Beschreibung der API selbst, welche als Grundlage für alle Operationen verwendet wird. Da sich der Prototyp beim Eingabeformat auf das OpenAPI Spezifikationsformat beschränkt, wurde zunächst ein OpenAPI Metamodell erstellt, welches in Abbildung 1 dargestellt ist. Dieses Modell wird von der Applikation als Basis-Datenstruktur verwendet. Zur Unterstützung mehrerer Eingabeformate müssten in einer späteren Iteration auch andere Spezifikationsformate untersucht werden, um Gemeinsamkeiten abzuleiten und ein Datenmodell zu erstellen in das alle Formate überführt werden können.

Es wurden ebenfalls zwei weitere Datenmodelle erstellt, die für die im Prototyp festgelegten Ziele benötigt werden. Zunächst wurde ein Modell der JSON-Schema Spezifikation erstellt, welche sich nur in wenigen Divergenzen von der OpenAPI Schema Definition unterscheidet:

1. Einige im OpenAPI Schema vorhandene Attribute wie bspw. `nullable` oder `deprecated` werden nicht unterstützt.
2. Die `type` Angabe kann sowohl ein String wie auch ein String-Array sein.
3. in `$schema` wird zusätzlich ein Link zur verwendeten JSON-Schema Version angegeben.

Diese Unterschiede müssen bei einer Umwandlung rekursiv aufgelöst werden, da ein Schema weitere verschaltete Schema-Definitionen enthalten kann. Das Modell einer JSON-Schema Definition ist in Abbildung 2 dargestellt.

Das Testsuite-Metamodell (Abbildung 3) stellt den Aufbau einer Testsuite dar, welche aus mehreren Testcases besteht. In jedem Testcase wird eine Anfrage mit entsprechenden Parametern ausgeführt, und eine Reihe von Assertions werden geprüft:

1. `ValidStatusCodeAssertion` prüft, ob der HTTP-Statuscode valide ist. Bei nominalen Testcases wird ein erfolgreicher Statuscode (2xx) erwartet, bei fehlerhaften Testcases ein Client-Fehler (4xx).

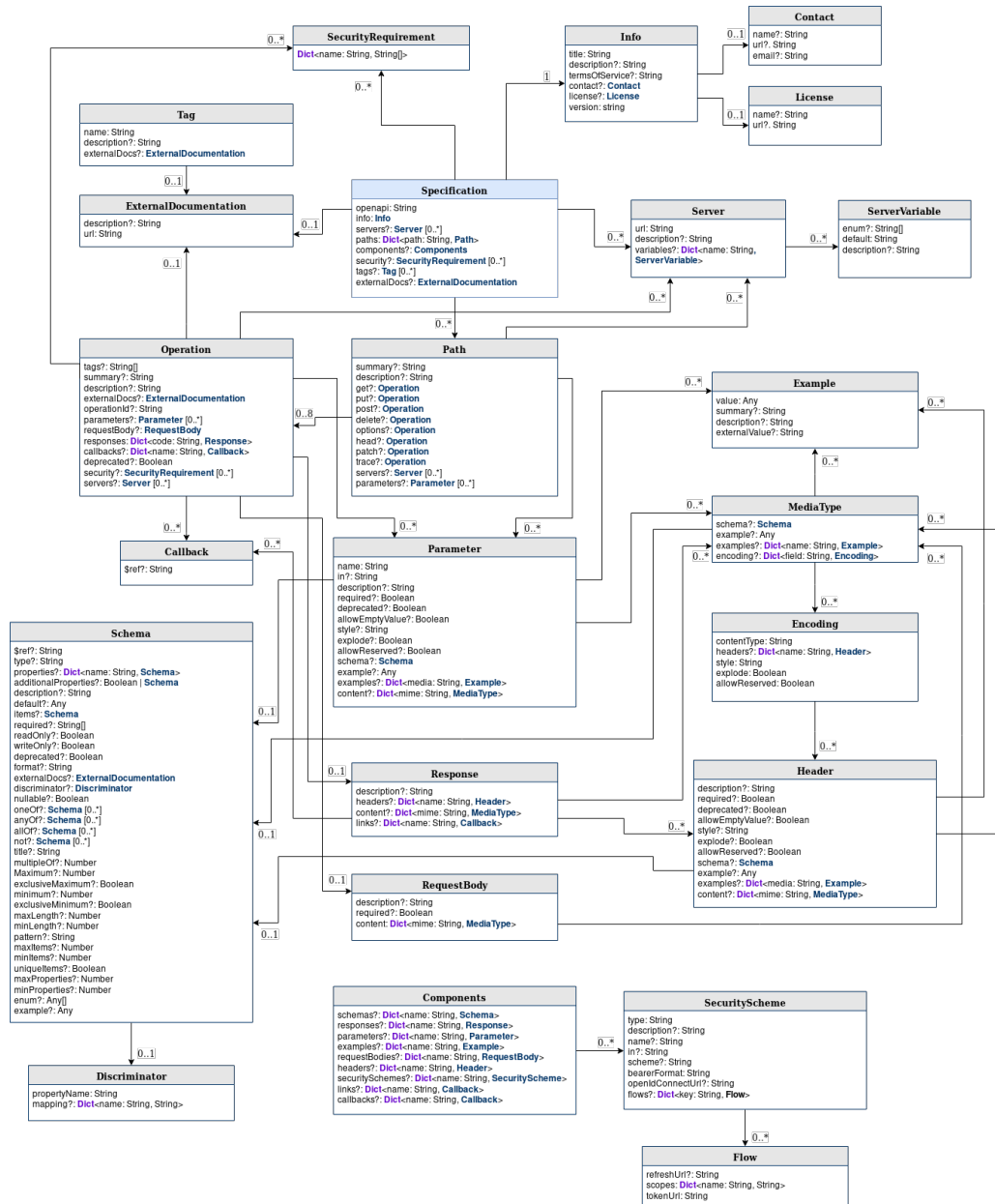


Abbildung 1: OpenAPI Metamodell

Schema
<pre> \$ref?: String type?: String String[] properties?: Dict<name: String, Schema> additionalProperties?: Boolean Schema description?: String default?: Any items?: Schema required?: String[] format?: String oneOf?: Schema [0..*] anyOf?: Schema [0..*] allOf?: Schema [0..*] not?: Schema [0..*] title?: String multipleOf?: Number Maximum?: Number exclusiveMaximum?: Boolean minimum?: Number exclusiveMinimum?: Boolean maxLength?: Number minLength?: Number pattern?: String maxItems?: Number minItems?: Number uniqueItems?: Boolean maxProperties?: Number minProperties?: Number enum?: Any[] </pre>

Abbildung 2: JSON-Schema Metamodell

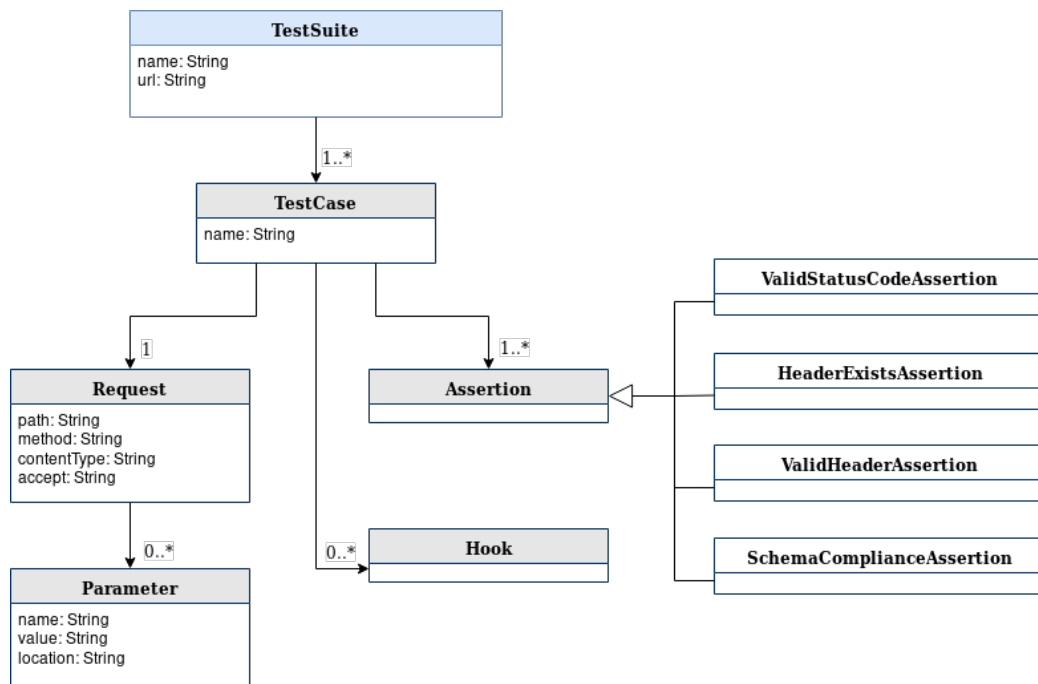


Abbildung 3: Testsuite Metamodell

2. `HeaderExistsAssertion` prüft, ob alle in der Spezifikation definierten HTTP-Header vorhanden sind.
3. `ValidHeaderAssertion` prüft, ob alle HTTP-Header valide Werte haben.
4. `SchemaComplianceAssertion` validiert die vom Server erhaltene Antwort mit dem passenden JSON-Schema.

Ein Testcase führt ebenfalls mehrere Hooks aus, die von dem Entwickler angepasst werden können um bspw. Authentifizierung zu ermöglichen.

1.2 Architektur

1.3 Vorgehensmodell