Inhaltsverzeichnis

1 Einführung in die Thematik

Innerhalb dieses einführenden Kapitels in die Thematik werden Grundlagen bezüglich der Bereiche, in denen sich das Projekt bewegt, aufgezeigt. Dies umfasst eine kurze Erläutering der Prinzipien von HTTP und REST, sowie eine Untersuchung der verschiedenen Deskriptionsformate zum Beschreiben von REST APIs.

1.1 HTTP

Das *Hypertext Transfer Protocol* (HTTP) ist ein Anwendungsprotokoll zur Kommunikation über ein Netzwerk. HTTP ist das primäre Kommunikationsprotokoll im World Wide Web, und wurde von der Internet Engineering Task Force (IETF) und dem World Wide Web Consortium (W3C) in einer Reihe von RFC-Dokumenten (z.B. *RFC 7230*, *RFC 7231* und *RFC 7540*) standardisiert. Eine HTTP-Nachricht wird in der Regel über TCP übertragen, und besteht aus 4 Komponenten:

Verb / Methode: Operationstyp, der durchgeführt werden soll, beispielsweise das Anfragen einer Ressource.

Ressourcenpfad: Ein Bezeichner, der angibt, auf welche Ressource die HTTP-Operation angewendet werden soll.

Headers: Zusätzliche Metadaten, ausgedrückt als Liste von Key/Value-Paaren.

Body: Nutzdaten der Nachricht.

Das HTTP Protokoll erlaubt die folgenden Operationen bzw. Methoden:

GET: Die angegebene Ressource sollte im Body-Teil der Antwort zurückgegeben werden.

HEAD: Wie *GET*, nur dass die Nutzdaten nicht zurückgegeben werden sollen. Dies ist nützlich, wenn nur überprüfen wird, ob eine Ressource existiert, oder wenn nur die Header abgerufen werden sollen.

POST: Daten werden an den Server geschickt. Häufig wird diese Methode benutzt, um neue Ressourcen anzulegen.

DELETE: Die angegebene Ressource löschen.

PUT: Die angegeben Ressource mit neuen Daten ersetzen.

PATCH: Ändert eine Ressource, ohne diese vollständig zu ersetzen.

TRACE: Liefert die Anfrage so zurück, wie der Server sie empfangen hat.

OPTIONS: Liefert eine Liste der vom Server unterstützten Methoden auf einer Ressource.

CONNECT: Eine Tunneling-Verbindung über einen HTTP-Proxy herstellen, die normalerweise für verschlüsselte Kommunikationen benötigt wird.

Wenn ein Client eine HTTP-Anfrage sendet, schickt der Server eine HTTP-Antwort mit Headern und möglicherweise Nutzdaten im Body zurück. Darüber hinaus enthält die Antwort auch einen numerischen, dreistelligen Statuscode. Es gibt fünf Gruppen von Statuscodes, die durch die erste Ziffer identifiziert werden können:

1xx: Wird für provisorische Antworten verwendet, wie z.B. der Information, dass die Bearbeitung der Anfrage trotz der Rückmeldung noch andauert.

2xx: Wird zurückgegeben, wenn die Anfrage erfolgreich bearbeitet wurde. Der Server könnte beispielsweise weiter spezifizieren, dass eine neue Ressource angelegt wurde (201), z.B. durch einen POST-Befehl, oder dass im Antwortkörper (204) nichts erwartet wird, z.B. durch einen DELETE-Befehl.

3xx: Wird wird Umleitungen benutzt. So kann dem Client bspw. mitgeteilt werden, dass sich eine Ressource an einem neuen Ort befindet.

4xx: Bei der Bearbeitung der Anfrage ist ein Fehler aufgetreten, der durch den Client verursacht wurde. Dies können beispielsweise falsch formatierte Anfragen (400), Anfragen die vom Server nicht bearbeitet werden können (422), oder auch einfach Anfragen auf nicht existierende Ressourcen (404) sein.

5xx: Bei der Bearbeitung der Anfrage ist ein Fehler aufgetreten, dessen Ursache beim Server liegt.

1.2 REST

Representational State Transfer (REST) ist eine Abstraktion der architektonischen Elemente innerhalb eines verteilten Hypermediasystems. Es ist somit ein Softwarearchitekturstil, der die architektonischen Prinzipien, Eigenschaften und Einschränkungen für die Umsetzung von internetbasierten verteilten Systemen definiert (Fielding und Taylor, 2000: 86). REST basiert auf fünf Kernprinzipien (Tilkov u. a., 2015: 11):

- Ressourcen als Abstraktion von Informationen, identifiziert durch einen eindeutigen *resource identifier*
- Verknüpfungen / Hypermedia
- Standardmethoden
- Unterschiedliche Repräsentationen
- Statuslose Kommunikation

In Folgendem wird untersucht, welche maschinenlesbare Spezifikationsformate zur Beschreibung von auf HTTP basierenden REST APIs zur Verfügung stehen. Ziel dieser Untersuchung ist es, eine abstrakte Modellierung von REST APIs zu konzipieren, die verwendet werden kann, um eine API Spezifikation in eine interne Datenstruktur zu überführen, welche dann anschließend angereichert wird um automatisiert Artefakte wie Dokumentation oder Test-Cases zu generieren. Im Zuge dessen wird ebenfalls analysiert, welche Möglichkeiten in diesem Feld bereits existieren.

1.3 Spezifikationsformate

1.3.1 Contract First und Contract Last

Es kann im Wesentlichen zwischen zwei verschiedenen Ansätzen zur Definition des Vertrages, den eine API garantiert, unterschieden werden (Spichale, 2017: 272):

- *Contract-First*. Bei diesem Ansatz wird zunächst der Vertrag (also die Spezifikation der API) geschrieben, und anschließend die Implementierung. Vorteilhaft ist hier der größere Fokus auf dem Design der Schnittstelle. In frühen Iterationen der Spezifikation kann in Abstimmung mit den Konsumenten der API mit geringen Aufwand der Vertrag gefestigt werden, sodass später bei der Implementierung weniger Änderungen anfallen. Nachteil ist, dass Entwickler mehr mit den eigentlichen Spezifikationen arbeiten müssen, und diese nicht automatisch generieren können.

- *Contract-Last*. Bei diesem Ansatz existiert die Implementierung bereits, und die Beschreibung wird nachträglich erzeugt. Häufig können hier unterstützend Tools benutzt werden, welche die Spezifikation automatisch aus der Implementierung generieren, und der Entwickler muss meistens nur noch einige Annotationen anpassen. Da bei der automatischen Generierung Schemas in vielen Fällen dupliziert werden, kann dies zu mehr Aufwand führen, wenn manuelle Anpassungen vorgenommen werden müssen. Ebenso reduziert dies die Wiederverwendbarkeit von Spezifikationsteilen oder Schemas (Zhong und Yang, 2009: 1).

Bei beiden Ansätzen müssen Tests angelegt werden, die sicherstellen, dass die Implementierung nicht von der Beschreibung bzw. Dokumentation abweicht.

1.3.2 RAML

RAML (*RESTful API Modeling Language*) ist ein auf YAML basierende Spezifikationsformat. Es wurde insbesondere für den *Contract-First* Ansatz konzipiert, der Schwerpunkt liegt damit auf dem API-Design (Spichale, 2017: 277; Tilkov u. a., 2015: 165). Beschreibungen können in Markdown formatiert

werden. Jedes RAML-Dokument beginnt mit der Angabe der RAML-Version und einigen allgemeinen Informationen zur API, wie Titel, URL und Version. Hier können ebenfalls Strukturierungshilfen wie Traits oder Datentypen definiert werden, welche dann im Rest des Dokumentes eingebunden werden können. Anschließend folgen die Beschreibungen der Ressourcen und Subressourcen. Zur Beschreibung von Requests und Responses können optional die bereits definierten Datentypen oder auch JSON-Schema Definitionen verwendet werden. Über *Includes* kann die Spezifikation in mehrere Dateien aufgeteilt werden.
Folgendes Beispiel wurde aus der RAML-Spezifikation entnommen (*RAML Version 1.0 Spezifikation* o.D.).

```
#%RAML 1.0
title: GitHub API
version: v3
baseUri: https://api.github.com
mediaType: application/json
securitySchemes:
  oauth_2_0: !include securitySchemes/oauth_2_0.raml
traits:
securedBy: [ oauth_2_0 ]
title: API with Types
types:
  User:
    type: object
    properties:
      firstname: string
      lastname: string
      age: number
/users/{id}:
  get:
    responses:
      200:
        body:
          application/json:
            type: User
```

Literatur

Fielding, Roy T und Richard N Taylor (2000). *Architectural styles and the design of network-based software architectures*. Bd. 7. University of California, Irvine Irvine, USA.

RAML Version 1.0 Spezifikation (o.D.). Abgerufen am 21. November 2018. URL: https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md.

Spichale, K. (2017). *API-Design: Praxishandbuch für Java- und Webservice-Entwickler*. dpunkt.verlag.

Tilkov, Stefan u. a. (2015). *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt. verlag.

Zhong, Youliang und Jian Yang (2009). „Contract-First design techniques for building enterprise web services“. In: *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, S. 591–598.