RESTful API mit node.js

AUTHOR

14. Februar 2018

Zusammenfassung

Dieses Dokument beschriebt die Theorie hinter REST[1] und die Implementierung einer RESTful API[2] mithilfe von node.js

[1] Representational State Transfer
[2] Application Programming Interface

Inhaltsverzeichnis

Listings

[3]Hypertext Transfer Protocol
[4]JavaScript Object Notation
[5]Extensible Markup Language

1 Allgemeines über REST

REST ist ein Programmierparadigma und wurde von Roy Fielding in seiner Dissertation[2] spezifiziert.

Die wichtigsten Eigenschaften[1, p. 2] sind:

- Alles ist eine Ressource und jede ist mit einer eindeutigen Adresse identifizierbar (URI[6])
- Verwendung von Standard-HTTP Methoden[7]
- Ressourcen können mehrerer Repräsentationen besitzen
- Zustandslose Kommunikation
- HATEOAS[8]

1.1 Ressourcen

Die Ressourcen sind unter einer eindeutigen Adresse (URI) erreichbar und können unter verschiedenen Repräsentationen dargestellt werden. Siehe Kapitel 1.3.

Zum Beispiel kann ein Ressource folgende URI haben "http://ipaddress:port/path/Ressource"

1.2 HTTP-Methoden

Die Verwendung von HTTP-Methoden ist keine strickte Voraussetzung für REST. In der Praxis wird jedoch hauptsächlich HTTP verwendet für REST. Deshalb wird hier auch nur auf die Implementierung mit HTTP-Methoden eingegangen.

Es sind folgende Operationen laut RFC2616[4] im HTTP-Protokoll möglich:

- GET
- POST
- PUT
- DELETE
- HEAD
- OPTIONS
- TRACE
- CONNECT

Für REST werden die folgenden 4 CRUD[9]-Operationen verwendet.

Create	POST
Read	GET
Update	PUT
Delete	DELETE

[6] Unique Adress Identifier
[7] Keine strenge Voraussetzung für REST, in der Praxis wird aber hauptsächlich HTTP verwendet
[8] Hypermedia as the Engine of Application State
[9] Create Read Update Delete

1.2.1 POST - Create

Erzeugt eine neue Ressource.

Beispiel:

Method	URI	Sent Data	Response
POST	/Persons	Ressource Data	/Persons/123

1.2.2 GET - Read

Um eine Ressource anzuzeigen wird die GET-Operation verwendet.

Beispiel:

Method	URI	Sent Data	Response
GET	/Persons/123	none	Resource Data

1.2.3 PUT - Update

Verändert eine bestehende Ressource.

Beispiel:

Method	URI	Sent Data	Response
PUT	/Persons/123	Ressource Data	none

1.2.4 DELETE - Delete

Löscht eine Ressource.

Beispiel:

Method	URI	Sent Data	Response
DELETE	/Persons/123	none	none

1.3 Repräsentation

Eine Ressource kann unterschiedliche Repräsentationen besitzen. Als Beispiel wird als eine Ressource eine Person betrachtet. Diese enthält folgende Daten:

- Vorname
- Nachname
- Telefonnummer

Die Person kann in verschiedenen Formaten dargestellt werden. In JSON würde die Repräsentation so aussehen:

```
1  {"vorname":"Hans","nachname":"Mueller","telefonnummer":"0680123456789"}
```

Listing 1: JSON-Repräsentation (application/json)

Die selbe Person kann allerdings auch in XML dargestellt werden:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <root>
3    <nachname>Mueller</nachname>
4    <telefonnummer>0680123456789</telefonnummer>
5    <vorname>Hans</vorname>
6  </root>
```

Listing 2: XML-Repräsentation (application/xml)

1.4 Zustandslose Kommunikation

Eine Vorraussetzung für eine RESTful API ist die Zustandslosigkeit. Weder Server noch Client sollten Zustandsinformationen speicher. Das heißt Anfragen an den Server sollten in sich geschlossen sein. Dies hat den Vorteil der einfachen Skalierbarkeit, das heißt Anfragen können an unterschiedliche Server verteilt werden. [1, p. 6]

1.5 Hypermedia as the Engine of Application State

HATEOAS ist laut Roy Fielding eine Voraussetzung für eine RESTful API.[3] Dazu müssen Ressourcen Links (Hypermedia) zu anderen Ressourcen enthalten können. Es sollte somit möglich sein nur durch Kenntnis der Root-URI zu allen möglichen URIs weiter zu navigieren. Somit kann man von einer Ressource zur Nächsten navigieren.
Oft werden jedoch auch fälschlicherweise APIs als RESTful bezeichnet die nicht HATEOAS verwenden.

2 Beispielprogramm

Da REST nur ein Programmierparadigma ist, hat man sehr viel Möglichkeiten bei der Implementierung. Programmiersprache, Repräsentation der Daten sowie Protokoll für die Datenübertragung sind nicht vorgegeben durch REST. Deshalb ist der erste Schritt für die Implementierung notwendig zu entscheiden welche Tools verwendet werden sollen.

2.1 Programmiersprache

Als Programmiersprache wird in diesem Beispiel Node.js verwendet.

> "Node.js® ist eine JavaScript-Laufzeitumgebung, die auf Chromes V8 JavaScript-Engine basiert. Durch ein Event-basiertes, blockierungsfreies I/O-Modell ist Node.js schlank und effizient."[10]

2.2 Daten Repräsentation

Da Ressourcen durch unterschiedliche Repräsentationen dargestellt werden können, könnten prinzipiell auch mehrere Repräsentationen vom Beispielprogramm unterstützt werden. Der Einfachheit halber wird allerdings nur ein Format verwendet. JSON bietet sich aufgrund seiner weiten Verbreitung und Kompatibilität mit JavaScript an. Doch JSON alleine bietet keine Verlinkungen zwischen Datensätzen an und erfüllt deshalb nicht HATEOAS. Deshalb gibt es zahlreiche Formate die auf JSON basieren aber zusätzlich auch HATEOAS unterstützen. [5]

- JSON-LD[11]
- HAL[12]
- Collection+JSON
- SIREN[13]

Für dieses Beispiel wurde Collection+JSON ausgewählt.

[10] https://nodejs.org/de/
[11] JSON for Linked Documents
[12] Hypertext Application Language
[13] Structured Interface for Representing Entities

2.2.1 Collection+JSON

Wenn beispielsweise die Root-URI aufgerufen wird, liefert der Server folgende Antwort.

```
{
  "collection": {
    "version": "1.0",
    "href": "http://localhost:1337",
    "items": [],
    "links": [
      {
        "rel": "home",
        "href": "http://localhost:1337",
        "promt": ""
      },
      {
        "rel": "persons",
        "href": "http://localhost:1337/persons",
        "promt": ""
      },
      {
        "rel": "adresses",
        "href": "http://localhost:1337/adresses",
        "promt": ""
      }
    ],
    "queries": [],
    "templates": []
  }
}
```

Listing 3: Collection+JSON-Repräsentation (application/vnd.collection+json)

Man sieht, dass man nun 2 Möglichkeiten hat weiter zu navigieren.
“http://localhost:1337/persons“ oder “http://localhost:1337/adresses“

2.3 Protokoll

Als Protokoll wird HTTP verwendet, weil es das am meisten verwendete für REST ist.

2.4 API Beschreibung

2.4.1 POST /persons

2.4.2 POST /addresses

2.4.3 GET /persons

2.4.4 GET /addresses

2.4.5 PUT /persons

2.4.6 PUT /addresses

2.4.7 DELETE /persons

2.4.8 DELETE /addresses

2.5 Source Code

Abkürzungen

REST	Representational State Transfer
API	Application Programming Interface
URI	Unique Adress Identifier
HTTP	Hypertext Transfer Protocol
HATEOAS	Hypermedia as the Engine of Application State
CRUD	Create Read Update Delete
JSON	JavaScript Object Notation
XML	Extensible Markup Language
JSON-LD	JSON for Linked Documents
HAL	Hypertext Application Language
SIREN	Structured Interface for Representing Entities

Literaturverzeichnis

[1] Valentin Bojinov. *RESTful Web API Design with Node.js*. Packt Publishing, 2015. ISBN 978-1-78398-586-9.

[2] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000. `https://www.ics.uci.edu/fielding/pubs/dissertation/fielding_dissertation_2up.pdf`, visited 2018-02-09.

[3] Roy Fielding. *REST APIs must be hypertext-driven!* 2008. URL `http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven`. visited 2018-02-09.

[4] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. Number 2616. RFC Editor, June 1999. URL `http://www.rfc-editor.org/rfc/rfc2616.txt`. visited 2018-02-09.

[5] Kevin Sookocheff. *On choosing a hypermedia type for your API - HAL, JSON-LD, Collection+JSON, SIREN, Oh My!* 2014. URL `https://sookocheff.com/post/api/on-choosing-a-hypermedia-format/`. visited 2018-02-09.