Representional State Transfer

Microservices

Msc. Informatik

13. April 2021

Warum REST?

- Genereller Standard für Webdienste
- Abstrahiert Implementierungsdetails
- Diverse Frameworks für REST-Anwendungen
- Konzept um Ressourcen oft geeignet für Webdienste
- Einfach zu dokumentieren
- Mit HATEOAS ohne Dokumentation nutzbar

Eckdaten

- wurde 2000 von Roy Thomas Fielding definiert
- parallel zu HTTP/1.1 entwickelt
- ist ein Architekturstil, kein Protokoll

Architekturprinzip

Null Style

Die REST Architektur ist eine Sammlung an Einschränkungen, die inkrementell auf ein System angewendet werden.
Der Null Style beschreibt hierbei einen leeren Satz von Einschränkungen.

Client-Server

Bedingung

- Trennung von Front- und Backend

Vorteile:

- Unabhängige Entwicklung beider Komponente
- Mehrere Frontends für ein Backend
- Bessere Skalierbarkeit

Stateless

Bedingung

- Es darf kein Sessionzustand auf dem Server gespeichert werden

Vorteile:

- Bessere Nachvollziehbarkeit von Requests
- Leichtere Erholung von Fehlschlägen
- Keine unnötige Ressourcenbelegung auf dem Server
- Vereinfachter Einsatz von Caches

Nachteil:

- Requests müssen immer alle nötigen Informationen enthalten -> Redundanz

Cache

Bedingung

- Daten müssen als (non-)cacheable gekennzeichnet werden

Vorteile:

- Reduzierte Netzwerklast
- Gefühlt schnellere Anwendung für den Nutzer

Nachteil:

- Cachedaten können veralten

Uniform Interface

Interface Einschränkungen

- Identifikation/Adressierbarkeit von Ressourcen
- Manipulation von Ressourcen durch deren Repräsentationen
- Selbstbeschreibende (HTTP) Nachrichten/Aktionen
- HATEOAS (Hypermedia as the engine of application state)

Vorteile:

- Trennung von Interface und Implementierung
- Nutzung ohne vertiefte Kenntnis

Nachteil:

- Mehraufwand durch Einhaltung der Einschränkungen

Layered System

Bedingung

- Komponente arbeiten auf mehreren Schichten

Vorteile:

- Austauschbarkeit einzelner Komponente
- Vereinfachung durch Schnittstellen

Nachteil:

- Erhöhte Latenzzeit

Code-On-Demand (optional)

Bedingung

- Einfach gehalten: Clients können code nachladen

Vorteile:

- Funktionalität kann nachgeliefert werden
- Server kann durch Auslagerung entlastet werden

Nachteil:

- Undurchsichtigkeit beim Nachladen

REST für ein Tasksystem

Ein kleines Tasksystem

Was die Anwendung können soll:

- Userverwaltung
- Taskverwaltung
- Kommunikation über HTTP
- Datenaustausch mit JSON

Models:

Task

- id
- userId
- title
- subject
- deadline

User

- id
- name
- email

Tasksystem: Client-Server

Client:

- Darstellung
- Userinteraktion
- Anfragenerstellung

Server:

- Bearbeitung von Anfragen
- Ressourcenverwaltung

Tasksystem: Stateless

Negativbeispiel

Um seine ihm zugewiesenen Tasks abzurufen, müssen folgenden Anfragen gemacht werden:

- GET /iAmUser?user=myUsername : Dem Server sagen, wer man ist
- GET /getMyTasks : Hier weiss der Server, wer der User ist

Stateless

Hier eine Variante, bei der der Server keine Informationen speichern muss:

- GET /tasks?user=myUsername

Tasksystem: Stateless find

Will man Items suchen, eignen sich zwei Typen von Request:

- GET: Informationen in URL
  - Route: /tasks/?title=<title>
- POST: Informationen im HTTP Body
  - Route: /tasks/
  - Body: 'title': '<title>'

Tasksystem: Uniform Interface

- Einstiegspunkt unter /
- Ressourcen:
  - User unter /users
  - Tasks unter /tasks
- Implementierung von GET, POST, PUT, DELETE für die Ressourcen
- HATEOAS: Links zwischen den einzelnen Routen

Tasksystem: HATEOAS

GET /user/1

```
{
  "id": 1,
  "name": "myUsername",
  "email": "test@example.org",
  "links": [{
    "name": "self",
    "href": "/user/1"
  }, {
    "name": "usertasks",
    "href": "/tasks?user=myUsername"
  }]
}
```

Tasksystem: Layered System

Die Anwendung muss in Schichten aufgebaut werden:

- Datenbanklayer
- Businesslogiclayer

Aber auch andere Mechanismen profitieren von einem Layered System:

- Load Balancer
- Cache (z.B. redis)

Tasksystem: Code-On-Demand

Hier kann Funktionalität beliebig nachgeladen werden:

- bootstrap über ein cdn
- JQuery u.A. JS-Frameworks