REST (Representational State Transfer)

Dissertation von Roy Fielding Architectural Styles and the Design of Network-based Software Architectures

Constraints

1. Client-Server
   - Trennung von User-Interface und Datenspeicher ⇒ Unabhängige Entwicklung von Client und Server.
2. Stateless
   - Jede Anfrage vom Client enthält alle benötigten Informationen. Server speichert keine Client-Session-Daten.
3. Cache
   - Client darf Daten speichern und muss für dieselben Daten nicht immer wieder neu anfragen.
4. Uniform Interface
   a) Resource identification in requests
      - Jede Ressource wird z.B. durch eindeutige URI identifiziert. URI ist Primärschlüssel der Ressource
      - Jede Ressource kann unterschiedliche Reräsentationen haben (z.B. XML, JSON).
   b) Resource manipulation through representations
      - Client kann über die Repräsentation die Ressource verändern oder löschen.
   c) Self-descriptive messages
      - Jede Nachricht enthält Information, wie die Daten zu parsen sind (z.B. XML, JSON).
   d) Hypermedia as the engine of application state (HATEOAS)
      - Zustandsautomat: Repräsentationen sind Zustände, Hyperlinks sind Zustandsübergänge
5. Layered System
   - Client erkennt keinen Unterschied zwischen Endserver und Zwischenservern für Loadbalancing und Caching.
6. Code-On-Demand (optional)
   - Server können ausführbaren Code an Client übertragen, z.B. JavaScript.

CRUD-Umsetzung mit HTTP

- GET: Daten lesen ohne Nebeneffekt (keine Zustandsänderung am Server).
- POST: Neue Subressource in übergeordneter Ressource erstellen (keine ID übergeben)
- PUT: Neue Ressource mit übergebener ID erstellen (bzw. ändern, falls sie schon existiert)
- DELETE: Ressource löschen