

REST API GUIDELINES

REST Überblick

- **R**epresentational **S**tate **T**ransfer
- Entwickelt von Roy Fielding als Teil seiner Dissertation (2000)
 - https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- **RESTful** Kernkomponenten
 - Resources (URIs) („Nouns“ : customers, articles, tickets)
 - HTTP Verbs (GET, PUT,...)
 - HTTP Status Codes (200, 201,...)
 - Stateless

REST Überblick

Resources (URIs)	HTTP Verbs w3.org/Protocols/rfc2616/rfc2616-sec9.html	HTTP Status Codes w3.org/Protocols/rfc2616/rfc2616-sec10.html
/Products /Products/87 /Customers /Orders/14 ...	GET PUT POST DELETE PATCH COPY HEAD OPTIONS CONNECT ...	200/OK 201/Created 202/Accepted 302/Found (redirect) 400/Bad Request 401/Unauthorised 404/Not Found 500/Server Error 504/Gateway Timeout ...

Microsoft

REST - Verben

Befehl (HTTP- Methode) ⇄	Beschreibung	⇄ Anmerkungen ⇄
GET	fordert die angegebene Ressource vom Server an. GET weist keine Nebeneffekte auf. Der Zustand am Server wird nicht verändert, weshalb GET als <i>sicher</i> bezeichnet wird.	[n 1]
POST	fügt eine neue (Sub-)Ressource unterhalb der angegebenen Ressource ein. Da die neue Ressource noch keinen URI besitzt, adressiert der URI die übergeordnete Ressource. Als Ergebnis wird der neue Ressourcenlink dem Client zurückgegeben. POST kann im weiteren Sinne auch dazu verwendet werden, Operationen abzubilden, die von keiner anderen Methode abgedeckt werden.	[n 2]
PUT	die angegebene Ressource wird angelegt. Wenn die Ressource bereits existiert, wird sie geändert.	[n 3]
PATCH ^[n 4]	ein Teil der angegebenen Ressource wird geändert. Hierbei sind Nebeneffekte erlaubt.	[n 5]
DELETE	löscht die angegebene Ressource.	[n 3]
HEAD	fordert Metadaten zu einer Ressource an.	[n 5][n 1]
OPTIONS	prüft, welche Methoden auf einer Ressource zur Verfügung stehen.	[n 5][n 1]
CONNECT	Dient dazu, die Anfrage durch einen TCP-Tunnel zu leiten. Wird meist eingesetzt, um eine HTTPS-Verbindung über einen HTTP-Proxy herzustellen.	[n 5][n 1][n 6]
TRACE	Gibt die Anfrage zurück, wie sie der Zielservice erhält. Dient etwa dazu, um Änderungen der Anfrage durch Proxyserver zu ermitteln.	[n 5][n 1][n 6]
<ol style="list-style-type: none"> 1. ↑ ^{a b c d e} nullipotent. Ein Aufruf dieser Methoden führt zu keinen Nebeneffekten. 2. ↑ nicht idempotent. Ein erneuter Aufruf erstellt für jeden Aufruf mit demselben URI ein neues Objekt, anstatt dasselbe Objekt zurückzugeben. 3. ↑ ^{a b} idempotent. Der erste Aufruf dieser Methoden mit einem bestimmten URI führt zu Nebeneffekten. Ein erneuter Aufruf mit demselben URI führt zu keinen weiteren Nebeneffekten. 4. ↑ siehe RFC 5789 5. ↑ ^{a b c d e} optional. Wird nicht für CRUD-Operationen benötigt. 6. ↑ ^{a b} Eine Implementierung dieser Methoden wirkt sich ggf. auf die Sicherheit der Anwendung aus. 		

https://de.wikipedia.org/wiki/Representational_State_Transfer

RESTful URIs und Actions

- Ressourcen sind die „**Nouns**“
 - customers, orders, tickets, groups
- HTTP-Verben sind die **Actions**
 - **GET /customers** – Liefert eine Liste von Kunden
 - **GET /customers/12** – Liefert einen spezifischen Kunden
 - **POST /customers** – Erzeugt eine neue Kunden-Ressource
 - **PUT /customers/12** – Ersetzt die Daten der Kunden-Ressource
 - **PATCH /customers/12** – Führt ein Teil-Update auf die Kunden-Ressource aus
 - **DELETE /customers/12** – Löscht* die Kunden-Ressource
- GET: verändert **niemals** Daten oder führt zu einem anderen Zustand der Ressource
- Plural für die „Nouns“ hat sich durchgesetzt, da es zu einem API-Endpunkt führt auf den die Actions ausgeführt werden können

RESTful URIs und Actions - Relationen

- **GET /customers/12/orders** – Liefert eine Liste von Aufträgen für den Kunden 12
- **GET /customers/12/orders/2** – Liefert den Auftrag mit der Id 2 für den Kunden 12
- **POST /customers/12/orders** – Erzeugt einen neuen Auftrag für den Kunden 12
- **PUT /customers/12/orders/2** – Ersetzt die Daten des Auftrags mit der Id 2
- **PATCH /customers/12/orders/2** – Führt ein Teil-Update auf den Auftrag aus
- **DELETE /customers/12/orders/2** – Löscht* den Auftrag mit der Id 2

RESTful URIs und Actions - Relationen

- Es wird empfohlen nicht tiefer als 2 Ebenen zu schachteln
 - z.B: customers/1/orders/123/orderitems
- Da REST beliebig viele Ressourcen erlaubt, ist das Problem durch einen eigenen Endpunkt leicht zu lösen
- orders/123/orderitems

RESTful URLs und Actions – Außerhalb von CRUD

- Manche Aktionen lassen sich nicht leicht in die RESTful-Struktur übersetzen
- Z.B. Favoritenmarkierung von Büchern
- Wichtig: Nicht in den RPC-Style zurück fallen
 - Don`t: **PUT /books/1/setasfavorite=true**

RESTful URLs und Actions – Außerhalb von CRUD

- Alternative 1: isFavorite als Feld in der Ressource aufnehmen und mit einem PUT aktualisieren
- Alternative 2:
 - **POST oder PUT /books/1/favorites**
 - **DELETE /books/1/favorites**

Dokumentation der API

- Gute APIs sind immer dokumentiert:
- Welche Möglichkeiten bietet die API?
- Was sind die Ressourcen?
- Zu welchem Ergebnis führen die HTTP-Verben wenn sie auf Ressourcen angewendet werden?
- Was sind die Änderungen zwischen Versionen? (Changelog)
- Wann werden alte API Versionen deaktiviert?
- <https://developer.github.com/v3/>

Versionierung der API

- Gute APIs sind immer versioniert
 - URIs sollten über einen großen Zeitraum hinweg gültig bleiben
 - Konsumenten der API können evtl. nicht sofort auf Änderungen der Schnittstelle reagieren
 - Alte Versionen können nach und nach abgeschaltet werden
- Möglichkeiten der Versionierung
 - Im HTTP-Header der Requests **Content Type: application/vnd.github.v3+json** oder **x-myApp-version: 2**
 - In der URL **api/v3/books**, **api/v3.0/books**
 - Als URI-Parameter **/api/books?v=2**
 - <https://developer.github.com/v3/>
 - Alle werden kontrovers diskutiert => Hauptsache es wird versioniert

Versionierung der API mit ASP.NET

- Von Hand über Routes
 - `[Route("api/v1/[controller]")]`
- Microsoft.AspNetCore.Mvc.Versioning
 - <https://github.com/microsoft/aspnet-api-versioning/wiki>
- Swagger Integration
 - Microsoft.AspNetCore.Mvc.Versioning.ApiExplorer
 - <https://github.com/microsoft/aspnet-api-versioning/tree/master/samples/aspnetcore/SwaggerSample>

Versionierung der API mit ASP.NET


```
services.AddApiVersioning(  
    options =>  
    {  
        options.ReportApiVersions = true;  
        options.AssumeDefaultVersionWhenUnspecified = true;  
    });
```

Versionierung der API mit ASP.NET

```
[ApiController]
[ApiVersion("1.0")]
[Route("api/[controller]")]
[Route("api/v{version:apiVersion}/[controller]")]
public class ValuesController : ControllerBase
```

```
[ApiController]
[ApiVersion("2.0")]
[Route("api/v{version:apiVersion}/[controller]")]
public class ValuesController : ControllerBase
```

Versionierung der API mit ASP.NET

 **swagger**

Select a spec V1

Sample API ^{1.0}

[/swagger/v1/swagger.json](#)

A sample application with Swagger, Swashbuckle, and API versioning.

[Terms of service](#)
[Contact Chuck Norris](#)
[MIT](#)

Values

GET

/api/Values

GET

/api/v1/Values


GET

/api/Values/{id}

GET

/api/v1/Values/{id}

Versionierung der API mit ASP.NET

 **swagger**

Select a spec V2

Sample API ^{2.0}

</swagger/v2/swagger.json>

A sample application with Swagger, Swashbuckle, and API versioning.

[Terms of service](#)

[Contact Chuck Norris](#)

[MIT](#)

Values

GET

/api/v2/Values

POST

/api/v2/Values

GET

/api/v2/Values/{id}

PUT

/api/v2/Values/{id}

DELETE

/api/v2/Values/{id}

Filtern, Suchen, Sortieren

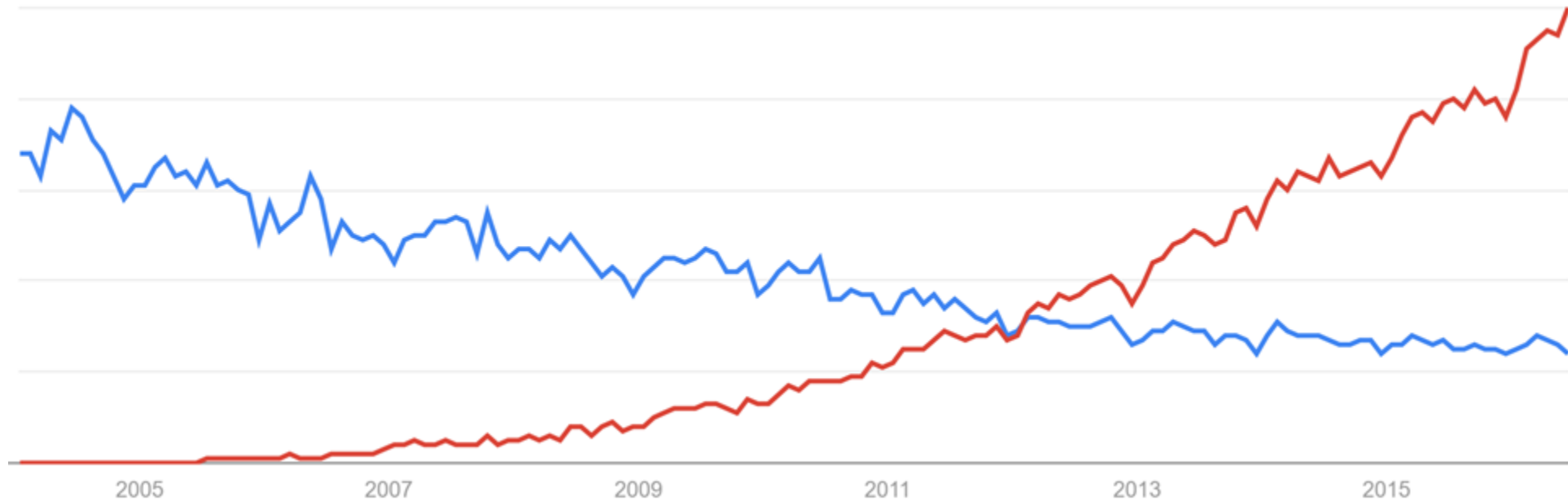
- Filterung
 - **GET /orders?state=completed**
- Sortierung
 - **GET /orders?sort=createdDate**
- Suchen
 - **GET /customers?q=Maier**
- Kombination
 - **GET /customers?q=Maier&state=new&sort=created,firstname**
 - Liefert Kunden deren Name **Maier** enthält, im Zustand **New** sind und sortiert diese nach **created** und **firstname**
- Best Practice:
 - Nicht mit den Filtermöglichkeiten übertreiben. Manchmal braucht man eine Suchtechnologie wie Lucene oder **ODATA (wird noch vorgestellt)**
 - Für häufige Queries sollten Aliase erstellt werden z.B. **GET /orders/recently_delivered**

Ressource nach Aktion wieder zurückgeben

- HTTP-Verben die Ressourcen erzeugen oder verändern sollten diese als Ergebnis an den Aufrufer zurückliefern
- **POST /customers** => liefert den neu erzeugten Kunden an den Aufrufer zurück mit **Status Code 201** und einem **Location-Header** der die URI auf den neuen Kunden enthält
- **PUT oder PATCH /customers/1** => liefert den veränderten Kunden an den Aufrufer zurück

XML oder JSON

- Viele Technologien unterstützen Content-Negotiation, für eine public-API sollte man sich aber heute (2022) auf JSON festlegen
- XML parsen deutlich rechenintensiver (Mobile)
- JSON benötigt deutlich weniger Datenvolumen



<http://www.google.com/trends/explore?q=xml+api#q=xml%20api%2C%20json%20api&cmpt=q>

HATEOAS (Hypermedia as the Engine of Application State)

- Teil des Dissertation von REST „Entdecker“ Roy Fiedling
- Ressourcen enthalten Links auf weiterführende Ressourcen und Aktivitäten

HATEOAS

GET /api/books

Das bisherige Result eines Requests auf den WebAPI Bookcontroller

```
[
  {
    "Id": 1,
    "Isbn": "1430242337",
    "Title": "C# Pro",
    "Price": 30,
    "Authors": [
      "Troelson"
    ],
    "ReleaseDate": "2013-06-24T11:11:07.6589397+02:00"
  },
  {
    "Id": 2,
    "Isbn": "161729134X",
    "Title": "C# in Depth",
    "Price": 40,
    "Authors": [
      "Skeet"
    ],
    "ReleaseDate": "2015-04-24T11:11:07.6589397+02:00"
  }
]
```

HATEOAS (Hypermedia as the Engine of Application State)

- Ziele
 - Weniger Roundtrips zum Server
 - Navigierbarkeit von Ressourcen

HATEOAS

HATEOAS-Style: Variante Links

Ressourcen erhalten eine Link-Collection. Diese gibt Auskunft, welche weiteren Aktionen auf dieser Ressource in ihrem aktuellen Zustand durchgeführt werden können

```
[
  {
    "id": 1,
    "isbn": "1430242337",
    "title": "C# Pro",
    "price": 30,
    "authors": [
      "Troelson"
    ],
    "releaseDate": "2013-06-24T11:42:09.6243749+02:00",
    "links": [
      {
        "rel": „self“,
        "href": "http://localhost:52360/api/books/1",
        "method": "GET"
      },
      {
        "rel": "Update",
        "href": "http://localhost:52360/api/books/1",
        "method": "PUT"
      },
      {
        "rel": "BookAuthors",
        "href": "http://localhost:52360/api/books/1/authors",
        "method": "GET"
      }
    ]
  }
]
```

HATEOAS mit ASP.NET Core

```
public class Link
{
    public string Rel { get; set; }
    public string Href { get; set; }
    public string Method { get; set; }
}
```

Link

- Rel: Verwendungszweck / Aktion
- Href: URL zur „Aktion“
- Method: zu verwendendes HTTP-Verb

HATEOAS Beispiele

- <https://api.github.com/users/florianwachs>
- https://github.com/florianwachs/FHRWebservices/tree/master/11_aspnetcorehateoas/lessons/AspNetCoreHateoasWithLinks

JSON Property Serialisierung

- **PascalCase**

- Ist untypisch und sollte nicht verwendet werden (leider Standardverhalten des JSONContractSerializers)
- FirstName

- **CamelCase**

- Häufig eingesetzt
- firstName

- **SnakeCase**

- Findet vermehrt bei neueren APIs Verwendung
- first_name

- Am besten eine JSON-Library verwenden die eine Modifizierung der Property-Serialisierung erlaubt

Pagination

- Bei zu vielen Ergebnissen über die API muss auf Pagination zurückgegriffen werden
- Häufig enthält die JSON-Response vom Server ein spezielles Paging-Element, welches die URI zur nächsten „Datenseite“ enthält. Bei Bedarf kann der Client diese dann aufrufen
- Moderne Alternative: Link-Header
 - <https://developer.github.com/v3/#pagination>
 - [Link Header Field https://tools.ietf.org/html/rfc5988#page-6](https://tools.ietf.org/html/rfc5988#page-6)

HTTP-Verb Override

- Manche Firmen lassen durch die Firewall nur GET und POST Requests
- Mit dem **HTTP-Header X-HTTP-Method-Override** kann dem Server trotzdem mitgeteilt werden, welche Methode eigentlich gemeint war
- Niemals mit GET-Requests machen, GET ändert nie den Zustand einer Ressource

```
POST /api/Person/4 HTTP/1.1
Host: localhost:10320
Content-Type: application/json
X-HTTP-Method-Override: PUT
Cache-Control: no-cache
```

<http://www.hanselman.com/blog/HTTPPUTOrDELETENotAllowedUseXHTTPMethodOverrideForYourRESTServiceWithASPNETWebAPI.aspx>
[X](#)

Rate Limit

- Besonders public APIs limitieren die Zugriffe der Clients
 - Zugriffe insgesamt
 - Zugriffe in einem bestimmten Zeitraum
- Wichtig ist, dem Client mitzuteilen, dass er ein Limit überschritten hat und wann er wieder auf die API zugreifen kann
- Häufig verwendete Response-Header:
 - **X-Rate-Limit-Limit**: Erlaubte Anzahl der Requests
 - **X-Rate-Limit-Remaining**: Noch verbleibende Menge an Requests
 - **X-Rate-Limit-Reset**: Sekunden bis das Limit zurückgesetzt wird

HTTP Status Codes

- Eine API sollte immer passende Status Codes zurückgeben
- **200 OK** - Response to a successful GET, PUT, PATCH or DELETE. Can also be used for a POST that doesn't result in a creation.
- **201 Created** - Response to a POST that results in a creation. Should be combined with a **Location header** pointing to the location of the new resource
- 204 No Content - Response to a successful request that won't be returning a body (like a DELETE request)
- 304 Not Modified - Used when HTTP caching headers are in play
- **400 Bad Request** - The request is malformed, such as if the body does not parse
- **401 Unauthorized** - When no or invalid authentication details are provided. Also useful to trigger an auth popup if the API is used from a browser
- 403 Forbidden - When authentication succeeded but authenticated user doesn't have access to the resource
- **404 Not Found** - When a non-existent resource is requested
- 405 Method Not Allowed - When an HTTP method is being requested that isn't allowed for the authenticated user
- 410 Gone - Indicates that the resource at this end point is no longer available. Useful as a blanket response for old API versions
- 415 Unsupported Media Type - If incorrect content type was provided as part of the request
- 422 Unprocessable Entity - Used for validation errors
- 429 Too Many Requests - When a request is rejected due to rate limiting
- **500 Internal Server Error**

<http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>

Zu Bedenken

■ Caching

- HTTP hat Caching eingebaut, mit Ansätzen wie ETag und Last-Modified können sie für APIs gut nutzbar gemacht werden
- https://en.wikipedia.org/wiki/HTTP_ETag
- <https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.29>

■ SSL

- APIs mit SSL verschlüsseln, um Inhalt der Kommunikation zu schützen und sicherzustellen dass der Inhalt nicht verändert wurde

■ Authentifizierung und Autorisierung

- APIs werden in den meisten Fällen durch Technologien wie OAUTH (2) geschützt (kommt noch in der Vorlesung)

Zu Bedenken

- Komprimierung
 - Datenpakete sollten auf jeden Fall vom Webserver mit Technologien wie gzip komprimiert werden. Extreme Einsparungsmöglichkeiten
- Fehlerbehandlung
 - Dem Client über Status Codes mitteilen wo die Ursache liegt. 4xx = Client-Error, 5xx Server-Error
 - Als Content der Response sollte eine verständliche Fehlermeldung im JSON-Format übertragen werden

Ressourcen

- https://de.wikipedia.org/wiki/Roy_Fielding
- <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>
 - Microsofts API Best-Practices
- <http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>
 - Hervorragende Zusammenfassung!
- <https://developer.github.com/>
 - Gutes Beispiel eine sehr gute Dokumentation für eine API
- <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>