REST API-Design

Best Practice - REST API-Design

Ein kleiner Überblick was derzeit die gängigen Standards bzgl. REST API-Codes sind. Gerne kann diese Dokumentation fortlaufend erweitert/geändert werden.

API-Methoden

Häufig genutzte REST-Methoden (auch: Verben) sind nachfolgend aufgelistet.

- **GET** > Beschreibt eine Abfrage bzgl. einer Ressource. Sofern vorhanden, wird die Ressource ausgegeben.
- **POST** > Es wird eine Ressource erstellt, dabei wird die Ressource als Payload mitgegeben.
- **PUT** > Eine Ressource wird *ersetzt*, z.B. eine vorhandene Entity Person wird als solche mit der mitgesendeten Payload ersetzt. Andernfalls wird ein *404* zurückgegeben, da die zu modifizierende Ressource nicht vorhanden ist.
- **PATCH** > Mit *PATCH* wird eine Ressource aktualisiert, z.B. das Attribut *name* einer Entity *Person*. Unterschied zu *PUT* ist, dass beim Patch nie die ganze Entity ersetzt wird!
- **DELETE** > Das löschen einer Ressource wird ermöglicht, Rückgabe ist ein StatusCode 204 bei erfolgreichem Aufruf.

> *https://stackoverflow.com/questions/28459418/rest-api-put-vs-patch-with-real-life-examples*

> *https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.1.2*

Idempotenz

Hintergrund

Idempotenz bedeutet für die Http-Methoden, dass diese nach demselben Aufruf immer das gleiche Ergebnis liefern.

Beispiel: > Ein Aufruf mit **`GET ../persons/12`** liefert immer die gleiche Response bei mehrfacher Abfrage, nämlich die Person mit ID 12.

oder

> *Ein Aufruf unter `PUT ../persons/13` liefert auch immer die gleich Response, nämlich die (neue) Person mit ID 13.*

oder

> *Ein Aufruf unter `DELETE ../persons/14` liefert immer einen StatusCode 204 bei mehrfachem identischen Aufrufen.*

D.h. idempotente REST Aufrufe dürfen bei mehrfachen Aufrufen die Resourse **nicht** modifizieren!

Überblick bzgl. der Idempotenz

Idempotenz

Yes	No
GET	POST
PUT	PATCH
DELETE	-

> *Idempotenz - https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html*

Offset-Pagination

Bei sehr großen Daten, die zurückgegeben werden, möchte man nicht alles auf einmal ausgeben. Offset-Pagination ermöglicht das "seitenweise" Abrufen einer Antwort . Zusätzlich ist das schonender für das System.

Beispiel:

Für eine Liste von Entities *Person* kann folgende URImit Parametern erstellt werden

```
GET ../persons?offset=0&limit=5
```

Die nächsten (fünf) Entities *Person* erhält man mit dem Ändern des `offset`

```
GET ../persons?offset=5&limit=5
```

> *Pagination - https://www.baeldung.com/rest-api-pagination-in-spring*

Weitere bekannte Methoden sind z.B. "Keyset Pagination" oder "Seek-Pagination" [1] [2] [3] [4]

Statuscode-Übersicht - Meistgenutzt

Ein kleiner Überblick der häufig genutzten HTTP-Statuscodes.

Code	String	Bedeutung
200	**`OK`**	Aufruf war erfolgreich, entsprechende Response mit Inhalt wird zurückgegeben.
201	**`Created`**	häufig bei POST/PUT > Ressource wurde erfolgreich erstellt, eventuell enthält der Header einen entsprechenden Link mit der neu erstellten Ressource.
204	**`No Content`**	Häufig bei Delete oder Patch Operationen, da ausgegangen wird, dass die aufrufende Seite genug Informationen über die Ressource hat(te).
206	**`Partial Content`**	Beim Paging nutzbar, wenn speziell teilweise Inhalte nur ausgegeben werden.

400	**`Bad Request`**	Die aufrufende Seite hat einen (Syntax-)Fehler beim Aufrufen der Ressource.
401	**`Unauthorized`**	Nicht authorisiert für den Zugriff der Ressource.
403	**`Forbidden`**	Zugriff zur Ressource ist verboten.
404	**`Not Found`**	Die abgefragte Ressource ist nicht gefunden worden.
410	**`Gone`**	Aufrufende Ressource war zwar vorhanden, wird aber nicht nach zurückgegeben, da sie (die Ressource) dauerhaft entfernt wurde und in naher Zukunft nicht mehr vorhanden sein wird.
422	**`Unprocessible Entity`**	Code deutet auf einen internen semantischen Fehler hin. Die Syntax ist korrekt, aber die Verarbeitung der Ressource kann trotzdem nicht vollzogen werden.
413	**`Payload Too Large`**	Wenn eine Rückgabe zu groß ist, bzw. man möchte die Rückgabe auf einen bestimmten Wert begrenzen.
416	**`Range Not Satisfiable`**	Sofern bei der Pagination ein falscher offset und/oder limit angegeben werden.

> *https://www.seoagentur.de/seo-handbuch/lexikon/s/statuscode-410-gone/*

Versionierung

Da Software einer Pflege bedarf und auch Änderungen bzw. Erweiterungen folgen folgen (können), ist eine Versionierung durchaus sinnvoll.

Dabei stellt sich die Frage, wann bei einer REST-Schnittstelle eine Version angepasst werden soll. Nachfolgend eine Auflistung:

- Daten in der *response* ändern sich
- Datentyp der *response* ändert sich
- Entfernung von Resources in der API

Sofern Endpunkte hinzugefügt oder Daten ergänzt werden, ist keine Anpassung der Version nötig, d.h. die aktuelle Version kann bleiben.

Möglichkeiten eine Versionierung einzusetzen, ist

- in der URI
- im Header als Parameter
  - custom key - value Konzept
  - als MIME Type
- adaptierend neue Endpunkte mit (eventuell neuen) DTOs

> *https://www.baeldung.com/rest-versioning*
>
> *https://dzone.com/articles/your-api-versioning-wrong*
>
> *https://nordicapis.com/introduction-to-api-versioning-best-practices/*
>
> *https://jsonapi.org/format/#status*

> https://dzone.com/articles/best-practice-api-versioning-for-http-rest-interfa