# 2.4 REST-APIs, HTTP-Methoden und CRUD

REST ist ein häufig verwendetes Prinzip für Web-APIs.

REST bedeutet:

> Representational State Transfer

REST-APIs stellen Ressourcen über eindeutige Adressen bereit und verwenden HTTP-Methoden, um Daten zu lesen, anzulegen, zu verändern oder zu löschen.

Typische Ressourcen sind:

- Benutzer
- Kunden
- Projekte
- Aufgaben
- Nachrichten
- Dateien

---

**Lernziele**

Nach dieser Seite solltest du erklären können:

- was eine REST-API ist
- was eine Ressource ist
- welche Aufgaben GET, POST, PUT, PATCH und DELETE besitzen
- was CRUD bedeutet
- wie HTTP-Methoden und CRUD zusammenhängen
- warum PUT und PATCH nicht dasselbe sind
- was Idempotenz bedeutet
- wie REST-Anfragen aufgebaut werden

---

**Ressourcen in einer REST-API**

Eine Ressource ist ein Objekt, das über die API verwaltet wird.

Beispiele:

| Ressource | Endpoint |
| --- | --- |
| Benutzer | `/users` |

| Ressource | Endpoint |
|---|---|
| Kunden | `/customers` |
| Projekte | `/projects` |
| Aufgaben | `/tasks` |
| Dateien | `/files` |

Eine einzelne Ressource wird häufig über eine ID angesprochen.

Beispiel:

```
/users/15
```

Bedeutung:

> Benutzer mit der ID 15

---

### HTTP-Methoden

Die HTTP-Methode beschreibt, welche Aktion mit einer Ressource ausgeführt werden soll.

| Methode | Aufgabe |
|---|---|
| GET | Daten abrufen |
| POST | neue Daten anlegen |
| PUT | Datensatz vollständig ersetzen |
| PATCH | einzelne Werte ändern |
| DELETE | Daten löschen |

Die Kombination aus Methode und Endpoint bestimmt die gewünschte Aktion.

Beispiel:

```
GET /users/15
```

Bedeutung:

>

> Benutzer 15 abrufen

Beispiel:

```
DELETE /users/15
```

Bedeutung:

> “ Benutzer 15 löschen

---

**GET – Daten lesen**

GET wird verwendet, um Daten abzurufen.

Beispiele:

```
GET /users
```

Alle Benutzer abrufen.

```
GET /users/15
```

Benutzer 15 abrufen.

```
GET /users?department=IT
```

Alle Benutzer aus der Abteilung IT abrufen.

Eine GET-Anfrage besitzt normalerweise keinen Request-Body.

Mögliche Antwort:

```
{
 "id": 15,
 "name": "Max Mustermann",
 "department": "IT",
```

```
  "active": true
}
```

> GET liest Daten und sollte keine Ressource verändern.

---

**POST – Daten anlegen**

POST wird häufig verwendet, um eine neue Ressource zu erstellen.

Beispiel:

```
POST /users
```

Request-Body:

```
{
  "name": "Max Mustermann",
  "department": "IT",
  "active": true
}
```

Mögliche Antwort:

```
Status: 201 Created

{
  "id": 15,
  "name": "Max Mustermann",
  "department": "IT",
  "active": true
}
```

Die ID wird häufig vom Server erzeugt.

> POST erstellt normalerweise eine neue Ressource.

---

**PUT – vollständig ersetzen**

PUT wird verwendet, um eine bestehende Ressource vollständig zu ersetzen.

Beispiel:

```
PUT /users/15
```

Request-Body:

```
{
  "name": "Max Mustermann",
  "department": "Support",
  "active": true
}
```

Bei PUT erwartet die API häufig den vollständigen Datensatz.

Wird ein vorhandenes Feld nicht übertragen, kann es abhängig von der API entfernt oder auf einen Standardwert gesetzt werden.

Beispiel:

Der bisherige Benutzer besitzt zusätzlich eine Telefonnummer.

Wenn diese im PUT-Request fehlt, könnte sie verloren gehen.

> PUT ersetzt normalerweise die vollständige Ressource.

---

**PATCH – einzelne Werte ändern**

PATCH wird verwendet, um nur bestimmte Felder einer Ressource zu verändern.

Beispiel:

```
PATCH /users/15
```

Request-Body:

```
{
  "department": "Support"
}
```

Nur die Abteilung wird geändert.

Andere Werte wie Name oder Aktivstatus bleiben erhalten.

> PATCH verändert einzelne Teile einer Ressource.

---

### PUT und PATCH im Vergleich

| Merkmal | PUT | PATCH |
|---|---|---|
| Änderung | vollständiger Datensatz | einzelne Felder |
| benötigte Daten | häufig alle Werte | nur geänderte Werte |
| Risiko | fehlende Werte können überschrieben werden | übrige Werte bleiben erhalten |
| Beispiel | komplettes Benutzerprofil ersetzen | nur Abteilung ändern |

Merksatz:

> PUT ersetzt vollständig. PATCH ändert teilweise.

---

### DELETE – Daten löschen

DELETE wird verwendet, um eine Ressource zu entfernen.

Beispiel:

```
DELETE /users/15
```

Mögliche Antwort:

```
Status: 204 No Content
```

Der Statuscode `204` bedeutet, dass die Anfrage erfolgreich war, aber kein zusätzlicher Inhalt zurückgegeben wird.

In Unternehmenssystemen werden Benutzer häufig nicht endgültig gelöscht, sondern deaktiviert.

Beispiel:

```
PATCH /users/15

{
  "active": false
}
```

Das kann sinnvoll sein, wenn Daten oder Protokolle erhalten bleiben müssen.

> DELETE entfernt eine Ressource. In der Praxis kann eine Deaktivierung sicherer sein.

---

**CRUD**

CRUD beschreibt vier grundlegende Datenoperationen.

CRUD bedeutet:

| Buchstabe | Bedeutung | Aufgabe |
|---|---|---|
| C | Create | erstellen |
| R | Read | lesen |
| U | Update | ändern |
| D | Delete | löschen |

Zuordnung zu HTTP-Methoden:

| CRUD | HTTP-Methode |
|---|---|
| Create | POST |
| Read | GET |
| Update | PUT oder PATCH |
| Delete | DELETE |

Merksatz:

> CRUD beschreibt die Datenoperation. Die HTTP-Methode beschreibt die technische Anfrage.

---

### Beispiel: Benutzerverwaltung

| Aufgabe | Methode | Endpoint |
|---|---|---|
| alle Benutzer abrufen | GET | `/users` |
| bestimmten Benutzer abrufen | GET | `/users/15` |
| neuen Benutzer anlegen | POST | `/users` |
| Benutzer vollständig ersetzen | PUT | `/users/15` |
| Benutzer teilweise ändern | PATCH | `/users/15` |
| Benutzer löschen | DELETE | `/users/15` |

---

### Sammlungen und einzelne Ressourcen

Ein Endpoint ohne ID bezeichnet häufig eine Sammlung.

Beispiel:

```
/users
```

Bedeutung:

> Sammlung aller Benutzer

Ein Endpoint mit ID bezeichnet eine einzelne Ressource.

Beispiel:

```
/users/15
```

Bedeutung:

> einzelner Benutzer mit der ID 15

Typische Verwendung:

| Anfrage | Bedeutung |
| --- | --- |
| `GET /users` | Sammlung abrufen |
| `POST /users` | neue Ressource zur Sammlung hinzufügen |
| `GET /users/15` | einzelne Ressource abrufen |
| `PATCH /users/15` | einzelne Ressource ändern |
| `DELETE /users/15` | einzelne Ressource löschen |

---

### Untergeordnete Ressourcen

Ressourcen können miteinander verbunden sein.

Beispiel:

```
GET /projects/20/tasks
```

Bedeutung:

> Alle Aufgaben des Projekts 20 abrufen

Einzelne Aufgabe:

```
GET /projects/20/tasks/5
```

Bedeutung:

> Aufgabe 5 aus Projekt 20 abrufen

Solche verschachtelten Endpoints sollten übersichtlich bleiben.

---

### Filter, Sortierung und Begrenzung

Query-Parameter können eine GET-Abfrage genauer bestimmen.

Filtern:

```
GET /users?active=true
```

Sortieren:

```
GET /users?sort=name
```

Begrenzen:

```
GET /users?limit=20
```

Mehrere Parameter:

```
GET /users?department=IT&active=true&limit=20
```

Typische Query-Parameter:

| Parameter | Aufgabe |
| --- | --- |
| `active=true` | aktive Benutzer filtern |
| `department=IT` | nach Abteilung filtern |
| `sort=name` | Ergebnisse sortieren |
| `limit=20` | Anzahl begrenzen |
| `page=2` | Ergebnisseite auswählen |

---

### Idempotenz

Idempotenz bedeutet:

> Eine Anfrage kann mehrfach ausgeführt werden, ohne dass sich das Ergebnis nach der ersten erfolgreichen Ausführung weiter verändert.

Beispiel:

```
DELETE /users/15
```

Nach dem ersten erfolgreichen Löschen ist der Benutzer entfernt.

Eine erneute identische Anfrage löscht nicht noch einen weiteren Benutzer.

Typische Einordnung:

| Methode | Normalerweise idempotent |
|---|---|
| GET | ja |
| PUT | ja |
| DELETE | ja |
| POST | nein |
| PATCH | abhängig von der Umsetzung |

POST ist normalerweise nicht idempotent.

Wird derselbe POST-Request mehrfach ausgeführt, könnten mehrere Benutzerkonten entstehen.

Beispiel:

```
POST /users
```

Bei zweimaliger Ausführung könnten zwei Datensätze angelegt werden.

---

**Doppelte Ausführungen verhindern**

Automatisierte Workflows können versehentlich mehrfach gestartet werden.

Mögliche Ursachen:

- Webhook wird erneut gesendet
- Netzwerkfehler verursacht einen Wiederholungsversuch
- Benutzer startet den Workflow mehrfach
- Zeitplan überschneidet sich
- Antwort des Zielsystems kommt zu spät

Mögliche Schutzmaßnahmen:

- eindeutige Mitarbeiter- oder Vorgangs-ID prüfen
- vor dem Erstellen nach vorhandenen Daten suchen
- Idempotency-Key verwenden

- Workflow-Ausführungen protokollieren
- doppelte Trigger erkennen
- POST-Anfragen nicht unkontrolliert wiederholen

Beispiel:

```
Prüfen, ob E-Mail-Adresse bereits vorhanden ist
                  |
          Benutzer vorhanden?
            /         \
           ja          nein
           |            |
           v            v
        abbrechen     Benutzer anlegen
```

---

### Typischer REST-Request

```
POST https://api.example.com/v1/users

Authorization: Bearer abc123
Content-Type: application/json

{
  "name": "Max Mustermann",
  "department": "IT"
}
```

Bestandteile:

| Bestandteil | Inhalt |
|---|---|
| Methode | POST |
| Endpoint | `/v1/users` |
| Authentifizierung | Bearer-Token |
| Datenformat | JSON |
| Body | neue Benutzerdaten |

---

### Typische Response

```
Status: 201 Created

{
  "id": 15,
  "name": "Max Mustermann",
  "department": "IT"
}
```

Der Server bestätigt, dass die Ressource erstellt wurde.

---

**REST und Zustandslosigkeit**

REST-APIs sind normalerweise zustandslos.

Das bedeutet:

> Jede Anfrage enthält alle Informationen, die der Server für ihre Verarbeitung benötigt.

Der Server sollte nicht darauf angewiesen sein, dass eine vorherige Anfrage bestimmte Informationen gespeichert hat.

Beispiel:

Jeder Request übermittelt erneut das benötigte Zugangstoken.

```
Authorization: Bearer abc123
```

Vorteile:

- Anfragen können unabhängig verarbeitet werden
- Systeme lassen sich leichter skalieren
- Fehler lassen sich besser eingrenzen
- mehrere Server können Anfragen bearbeiten

---

**Sicherheit**

Besonders verändernde Methoden müssen geschützt werden.

Kritische Methoden:

- POST
- PUT
- PATCH
- DELETE

Wichtige Maßnahmen:

- HTTPS verwenden
- Benutzer authentifizieren
- Berechtigungen prüfen
- Eingabedaten validieren
- Least Privilege verwenden
- Änderungen protokollieren
- Löschvorgänge absichern
- Tokens nicht im Klartext speichern
- Test- und Produktivsystem trennen

Ein gültiges Token bedeutet nicht automatisch, dass jede Aktion erlaubt sein darf.

---

**Praxisbeispiel mit n8n**

Ein neuer Mitarbeiter soll im SaaS-System angelegt werden.

Ablauf:

1. n8n erhält die Mitarbeiterdaten.
2. Die Daten werden geprüft.
3. Mit GET wird nach der E-Mail-Adresse gesucht.
4. Ist kein Benutzer vorhanden, wird POST verwendet.
5. Die Response wird geprüft.
6. Die neue Benutzer-ID wird gespeichert.
7. Mit PATCH können weitere Eigenschaften ergänzt werden.
8. Das Ergebnis wird protokolliert.

Darstellung:

```
Trigger
  |
  v
GET /users?email=...
  |
  v
Benutzer vorhanden?
  /      \
```

```
 ja          nein
 |           |
 v           v
stoppen     POST /users
              |
              v
          Response prüfen
              |
              v
          Ergebnis speichern
```

---

### Systematische Fehlersuche

Wenn eine REST-Anfrage fehlschlägt:

1. Ist die HTTP-Methode korrekt?
2. Ist der Endpoint korrekt?
3. Wird eine Sammlung oder einzelne Ressource angesprochen?
4. Sind Path- und Query-Parameter korrekt?
5. Ist der JSON-Body vollständig?
6. Wird PUT oder PATCH richtig verwendet?
7. Ist die Ressource bereits vorhanden?
8. Ist die Authentifizierung gültig?
9. Sind ausreichende Berechtigungen vorhanden?
10. Welcher HTTP-Statuscode wird zurückgegeben?
11. Enthält die Response eine Fehlermeldung?
12. Kann die Anfrage gefahrlos erneut ausgeführt werden?

---

### Wichtige Begriffe

| Begriff | Bedeutung |
|---|---|
| REST | Prinzip zur Gestaltung von Web-APIs |
| Ressource | verwaltetes Objekt |
| Sammlung | Gruppe mehrerer Ressourcen |
| GET | Daten abrufen |
| POST | Daten erstellen |
| PUT | Ressource vollständig ersetzen |
| PATCH | einzelne Werte ändern |
| DELETE | Ressource löschen |

| Begriff | Bedeutung |
| --- | --- |
| CRUD | Create, Read, Update und Delete |
| Idempotenz | wiederholte Anfrage verändert das Ergebnis nicht weiter |
| Query-Parameter | filtern oder verändern eine Abfrage |
| Zustandslosigkeit | jeder Request enthält alle benötigten Informationen |

---

### Gesamtmerksatz

> REST-APIs verwalten Ressourcen über Endpoints. GET liest Daten, POST erstellt neue Daten, PUT ersetzt vollständige Datensätze, PATCH verändert einzelne Werte und DELETE entfernt Ressourcen. CRUD fasst diese grundlegenden Datenoperationen zusammen.

---

### Kontrollfragen

**Was bedeutet REST?**

Representational State Transfer.

**Was ist eine Ressource?**

Ein Objekt, das über eine API verwaltet wird, beispielsweise ein Benutzer oder Projekt.

**Welche Aufgabe besitzt GET?**

Daten abrufen.

**Welche Aufgabe besitzt POST?**

Neue Daten beziehungsweise Ressourcen erstellen.

**Was ist der Unterschied zwischen PUT und PATCH?**

PUT ersetzt normalerweise die vollständige Ressource. PATCH verändert nur einzelne Werte.

**Was bedeutet CRUD?**

Create, Read, Update und Delete.

**Welche HTTP-Methode ist normalerweise nicht idempotent?**

POST.

**Warum können doppelte POST-Anfragen problematisch sein?**

Weil dadurch mehrere identische Ressourcen erstellt werden können.

**Warum sollte vor dem Anlegen eines Benutzers zunächst gesucht werden?**

Damit kein doppeltes Benutzerkonto entsteht.

---

**Quellen**


- IETF – RFC 9110: HTTP Semantics
- MDN Web Docs – HTTP Request Methods
- MDN Web Docs – GET
- MDN Web Docs – POST
- MDN Web Docs – PUT
- MDN Web Docs – PATCH
- MDN Web Docs – DELETE


---