2.6 JSON – Aufbau und Verarbeitung strukturierter Daten

JSON ist ein häufig verwendetes Datenformat für APIs und Automatisierungen.

JSON bedeutet:

> JavaScript Object Notation

JSON ist textbasiert und kann von Menschen gelesen sowie von Programmen verarbeitet werden.

Typische Einsatzbereiche:

- API-Requests und Responses
- Webhooks
- n8n-Workflows
- Konfigurationsdateien
- Datenaustausch zwischen SaaS-Systemen
- Speicherung strukturierter Informationen

Lernziele

Nach dieser Seite solltest du erklären können:

- was JSON ist
- wie Objekte und Arrays aufgebaut sind
- welche Datentypen JSON unterstützt
- was Schlüssel-Wert-Paare sind
- wie verschachtelte Daten gelesen werden
- worin sich `null`, leere Werte und fehlende Felder unterscheiden
- wie JSON in APIs und n8n verwendet wird
- wie JSON-Fehler erkannt werden

Grundaufbau

JSON speichert Daten als Schlüssel-Wert-Paare.

Beispiel:

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

Die Schlüssel sind:

- `name`
- `department`
- `active`

Die zugehörigen Werte sind:

- `"Max Mustermann"`
- `"IT"`
- `true`

> Ein Schlüssel beschreibt eine Information. Der Wert enthält die dazugehörigen Daten.

JSON-Objekt

Ein JSON-Objekt wird mit geschweiften Klammern dargestellt:

```
{
 "name": "Max Mustermann",
 "email": "max.mustermann@example.com"
}
```

Ein Objekt enthält:

- Schlüssel
- Doppelpunkt
- Wert
- Komma zwischen mehreren Einträgen

Grundaufbau:

```
{
  "schluessel": "wert"
}
```

Mehrere Werte:

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

Schlüssel

JSON-Schlüssel müssen in doppelten Anführungszeichen stehen.

Richtig:

```
{
  "name": "Max Mustermann"
}
```

Falsch:

```
{
  name: "Max Mustermann"
}
```

Falsch:

```
{
  'name': 'Max Mustermann'
}
```

JSON verwendet doppelte Anführungszeichen.

> Schlüssel und Textwerte stehen in JSON in doppelten Anführungszeichen.

Unterstützte Datentypen

JSON unterstützt folgende Datentypen:

Datentyp	Beispiel
String	`"Max Mustermann"`
Zahl	`25`
Boolean	`true` oder `false`
Objekt	`{ "department": "IT" }`
Array	`["Slack", "Zoho", "Asana"]`
Null	`null`

JSON besitzt keinen eigenen Datentyp für:

- Datum
- Uhrzeit
- Datei
- Kommentar
- Funktion
- undefinierten Wert

Solche Werte werden meistens als Text übertragen.

Beispiel für ein Datum:

```
{
 "startDate": "2026-08-03"
}
```

String

Ein String ist ein Textwert.

Beispiel:

```
{
 "name": "Max Mustermann"
}
```

Auch Zahlen können als Text gespeichert werden:

```
{
 "employeeNumber": "0015"
}
```

Die Anführungszeichen sind wichtig.

Unterschied:

```
{
 "value": 15
}
```

`15` ist eine Zahl.

```
{
 "value": "15"
}
```

`"15"` ist Text.

Zahl

Zahlen stehen ohne Anführungszeichen.

Beispiel:

```
{
 "age": 30,
 "licenseCount": 25,
 "price": 19.99
}
```

Mit Zahlen können Programme Berechnungen durchführen.

Eine als String gespeicherte Zahl muss möglicherweise zuerst umgewandelt werden.

Boolean

Ein Boolean besitzt nur zwei mögliche Werte:

- `true`
- `false`

Beispiel:

```
{
 "active": true,
 "administrator": false
}
```

Boolean-Werte stehen ohne Anführungszeichen.

Richtig:

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

Falsch:

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

`"true"` wäre ein Textwert und kein Boolean.

Null

`null` bedeutet:

> Für dieses Feld ist aktuell kein Wert vorhanden.

Beispiel:

```
{
  "phone": null
}
```

`null` ist nicht dasselbe wie ein leerer String.

Wert	Bedeutung
`null`	kein Wert vorhanden
`""`	leerer Text
`0`	Zahlenwert null
`false`	Boolean-Wert falsch
Feld fehlt	Information wurde nicht übertragen

Beispiel:

```
{
  "phone": null
}
```

Das Feld ist vorhanden, besitzt aber keinen Wert.

Beispiel:

```
{
  "phone": ""
}
```

Das Feld enthält einen leeren Text.

JSON-Array

Ein Array ist eine geordnete Liste von Werten.

Arrays werden mit eckigen Klammern dargestellt.

Beispiel:

```
{
 "systems": [
  "Google Workspace",
  "Slack",
  "Zoho CRM"
 ]
}
```

Ein Array kann enthalten:

- Texte
- Zahlen
- Boolean-Werte
- Objekte
- weitere Arrays

> Ein Objekt enthält benannte Eigenschaften. Ein Array enthält eine Liste von Elementen.

Array mit Objekten

APIs übertragen häufig mehrere Datensätze als Array.

Beispiel:

```
{
 "users": [
  {
   "id": 15,
   "name": "Max Mustermann"
  },
  {
   "id": 16,
   "name": "Anna Beispiel"
  }
 ]
}
```

Das Array `users` enthält zwei Benutzerobjekte.

Verschachtelte Objekte

Ein JSON-Objekt kann weitere Objekte enthalten.

Beispiel:

```
{
  "name": "Max Mustermann",
  "department": {
    "id": 3,
    "name": "IT"
  }
}
```

Das Objekt `department` enthält:

- `id`
- `name`

Zugriffspfad:

```
department.name
```

Ergebnis:

```
IT
```

Objekte und Arrays kombiniert

JSON-Daten können komplex verschachtelt sein.

Beispiel:

```
{
  "name": "Max Mustermann",
  "department": "IT",
  "roles": [
    "support",
    "user"
```

```
  ],
  "devices": [
    {
      "type": "Laptop",
      "status": "assigned"
    },
    {
      "type": "Smartphone",
      "status": "ordered"
    }
  ]
}
```

Enthalten sind:

- Strings
- ein Array mit Rollen
- ein Array mit Geräteobjekten

Kommas und Klammern

Mehrere Einträge werden durch Kommas getrennt.

Richtig:

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

Falsch:

```
{
  "name": "Max Mustermann"
  "active": true
}
```

Nach dem letzten Eintrag darf in strengem JSON kein Komma stehen.

Falsch:

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

Klammern müssen korrekt geschlossen werden:

- `{ }` für Objekte
- `[ ]` für Arrays

Kommentare

JSON unterstützt keine Kommentare.

Falsch:

```
{
  "active": true,
  // Benutzer ist aktiviert
  "department": "IT"
}
```

Zusätzliche Hinweise müssen als eigenes Feld gespeichert werden:

```
{
  "active": true,
  "comment": "Benutzer ist aktiviert",
  "department": "IT"
}
```

Sonderzeichen

Sonderzeichen können mit einem Backslash maskiert werden.

Beispiele:

Zeichen	Schreibweise
Anführungszeichen	`\"`
Backslash	`\\`

Zeichen	Schreibweise
Zeilenumbruch	`\n`
Tabulator	`\t`

Beispiel:

```
{
  "message": "Der Benutzer \"Max Mustermann\" wurde angelegt."
}
```

JSON in einem API-Request

Ein neuer Benutzer soll angelegt werden.

Request:

```
POST /users

Content-Type: application/json

{
  "name": "Max Mustermann",
  "email": "max.mustermann@example.com",
  "department": "IT",
  "active": true
}
```

Der Header teilt dem Server mit:

> Der Request-Body enthält JSON-Daten.

JSON in einer API-Response

Mögliche Antwort:

```
Status: 201 Created
```

```
{
  "id": 15,
  "name": "Max Mustermann",
  "email": "max.mustermann@example.com",
  "department": "IT",
  "active": true
}
```

Die API ergänzt hier die neue Benutzer-ID.

Erfolgs- und Fehlerantwort

Erfolgreiche Response:

```
{
  "success": true,
  "userId": 15
}
```

Fehlerhafte Response:

```
{
  "success": false,
  "error": {
    "code": "EMAIL_ALREADY_EXISTS",
    "message": "Die E-Mail-Adresse ist bereits vorhanden."
  }
}
```

Ein Workflow sollte nicht nur den Statuscode, sondern auch die enthaltenen Daten auswerten.

Datenmapping

Datenmapping bedeutet:

> Ein Wert aus einem System wird einem passenden Feld in einem anderen System zugeordnet.

Beispiel:

Personalsystem	Google Workspace
`firstName`	`givenName`
`lastName`	`familyName`
`mail`	`primaryEmail`
`department`	`organization.department`

Beispieldaten aus dem Quellsystem:

```
{
  "firstName": "Max",
  "lastName": "Mustermann",
  "mail": "max.mustermann@example.com"
}
```

Benötigte Struktur im Zielsystem:

```
{
  "givenName": "Max",
  "familyName": "Mustermann",
  "primaryEmail": "max.mustermann@example.com"
}
```

Die Werte bleiben gleich, aber die Feldnamen unterscheiden sich.

JSON in n8n

In n8n werden Daten zwischen Nodes meistens als JSON-Objekte übertragen.

Beispiel:

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

Auf einzelne Werte kann über Ausdrücke zugegriffen werden.

Beispiel:

```
{{$json.name}}
```

Ergebnis:

```
Max Mustermann
```

Beispiel:

```
{{$json.department}}
```

Ergebnis:

```
IT
```

Bei verschachtelten Daten:

```
{{$json.department.name}}
```

Mehrere Datensätze in n8n

n8n verarbeitet häufig mehrere Items.

Beispiel:

```
[
  {
    "name": "Max Mustermann",
    "department": "IT"
  },
  {
    "name": "Anna Beispiel",
    "department": "Vertrieb"
  }
]
```

Jedes Objekt kann als eigenes Workflow-Item verarbeitet werden.

Beispiel:

- für jeden Benutzer ein Konto anlegen
- für jeden Kunden ein Projekt erstellen
- für jede Rechnung einen Prüfprozess starten

Datenvalidierung

Vor der Verarbeitung sollten JSON-Daten geprüft werden.

Beispiel für Pflichtfelder:

- Name vorhanden
- E-Mail-Adresse vorhanden
- Abteilung vorhanden
- Startdatum gültig
- Aktivstatus besitzt Boolean-Wert

Beispielablauf:

Schema

Ein JSON-Schema kann beschreiben:

- welche Felder erlaubt sind
- welche Felder verpflichtend sind

- welche Datentypen erwartet werden
- welche Werte zulässig sind
- wie Objekte verschachtelt sind

Beispielanforderung:

Feld	Datentyp	Pflichtfeld
`name`	String	ja
`email`	String	ja
`department`	String	ja
`active`	Boolean	nein

Ein Schema hilft dabei, fehlerhafte Daten frühzeitig zu erkennen.

Typische JSON-Fehler

Häufige Fehler:

- einfache statt doppelte Anführungszeichen
- fehlendes Komma
- zusätzliches Komma am Ende
- nicht geschlossene Klammer
- Schlüssel ohne Anführungszeichen
- ungültiger Boolean-Wert
- Kommentar im JSON
- falscher Datentyp
- ungültiges Sonderzeichen
- falsche Verschachtelung

Fehlerbeispiel

Ungültig:

```
{
 name: 'Max Mustermann',
 "active": "true",
}
```

Fehler:

- Schlüssel `name` besitzt keine Anführungszeichen
- einfache Anführungszeichen werden verwendet

- Boolean wurde als String gespeichert
- zusätzliches Komma am Ende

Korrekt:

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

Falscher Datentyp

Eine API erwartet:

```
{
 "licenseCount": 15
}
```

Gesendet wird:

```
{
 "licenseCount": "fünfzehn"
}
```

Die JSON-Syntax ist gültig, aber der Inhalt besitzt den falschen Datentyp.

Mögliche Response:

```
Status: 422 Unprocessable Content
```

> Gültiges JSON bedeutet nicht automatisch gültige Fachdaten.

Fehlendes und leeres Feld

Beispiel mit leerem Wert:

```
{
 "email": ""
}
```

Beispiel mit `null`:

```
{
 "email": null
}
```

Beispiel ohne Feld:

```
{
 "name": "Max Mustermann"
}
```

Diese Varianten können von einer API unterschiedlich behandelt werden.

Die API-Dokumentation muss deshalb geprüft werden.

Sicherheit und Datenschutz

JSON kann vertrauliche Daten enthalten.

Beispiele:

- personenbezogene Daten
- Kundendaten
- Zugangstoken
- interne IDs
- Zahlungsinformationen

JSON-Daten sollten daher:

- nur verschlüsselt übertragen werden
- nur benötigte Felder enthalten
- nicht unkontrolliert protokolliert werden
- nicht öffentlich gespeichert werden
- nach dem Least-Privilege-Prinzip verarbeitet werden

Nicht vollständig protokollieren:

```
{
  "password": "geheimes-passwort",
  "accessToken": "vollstaendiges-token"
}
```

Geheimnisse sollten maskiert werden:

```
{
  "accessToken": "***"
}
```

Systematische Fehlersuche

Wenn JSON nicht verarbeitet wird:

1. Sind alle Klammern korrekt geschlossen?
2. Werden doppelte Anführungszeichen verwendet?
3. Sind alle Einträge durch Kommas getrennt?
4. Befindet sich nach dem letzten Eintrag ein zusätzliches Komma?
5. Sind die Datentypen korrekt?
6. Sind alle Pflichtfelder vorhanden?
7. Entspricht die Verschachtelung der API-Dokumentation?
8. Ist der Header `Content-Type: application/json` gesetzt?
9. Enthält die Response eine genaue Fehlermeldung?
10. Ist das JSON syntaktisch gültig, aber fachlich falsch?

Wichtige Begriffe

Begriff	Bedeutung
JSON	textbasiertes Format für strukturierte Daten
Objekt	Sammlung von Schlüssel-Wert-Paaren
Array	geordnete Liste von Werten
Schlüssel	Name einer Information
Wert	Inhalt eines Feldes
String	Textwert
Boolean	`true` oder `false`
Null	kein vorhandener Wert
Verschachtelung	Objekt oder Array innerhalb eines anderen Objekts

Begriff	Bedeutung
Datenmapping	Zuordnung von Feldern zwischen Systemen
Validierung	Prüfung von Struktur und Inhalt
JSON-Schema	Beschreibung der erwarteten Datenstruktur

Gesamtmerksatz

> JSON überträgt strukturierte Daten als Objekte, Arrays und Schlüssel-Wert-Paare. APIs und Automatisierungsplattformen verwenden JSON für Requests, Responses und Webhooks. Neben der korrekten Syntax müssen auch Datentypen, Pflichtfelder, Verschachtelung und Datenschutz geprüft werden.

Kontrollfragen

Was bedeutet JSON?

JavaScript Object Notation.

Wie wird ein JSON-Objekt dargestellt?

Mit geschweiften Klammern.

Wie wird ein JSON-Array dargestellt?

Mit eckigen Klammern.

Welche Anführungszeichen verwendet JSON?

Doppelte Anführungszeichen.

Was ist ein Schlüssel-Wert-Paar?

Ein Schlüssel beschreibt eine Information und der Wert enthält die dazugehörigen Daten.

Was ist der Unterschied zwischen `true` und `"true"`?

`true` ist ein Boolean. `"true"` ist ein String.

Was bedeutet `null`?

Für das Feld ist aktuell kein Wert vorhanden.

Was bedeutet Datenmapping?

Felder eines Quellsystems werden passenden Feldern eines Zielsystems zugeordnet.

Warum kann syntaktisch gültiges JSON trotzdem abgelehnt werden?

Weil Pflichtfelder, Datentypen oder fachliche Werte falsch sein können.

Warum sollten Tokens nicht vollständig protokolliert werden?

Weil sie vertrauliche Zugangsdaten darstellen.

Quellen

- IETF – RFC 8259: The JavaScript Object Notation Data Interchange Format
- MDN Web Docs – JSON
- JSON Schema
- n8n-Dokumentation – Data Structure
- n8n-Dokumentation – Expressions