

Inhalt

Aufgabe 1a: Recherche zu REST APIs.....	2
Frage 1: Was ist eine REST API?	2
Frage 2: Was bewirken die unterschiedlichen HTTP-Verben für eine REST API?	2
Frage 3: Gibt es weitere HTTP-Verben? Wenn ja, beschreibe sie ausführlich.....	3
Frage 4: Wie sehen typische REST-Anfragen und Antworten aus?.....	4
Frage 5: Wie können REST-Anfragen in JavaScript mit der Fetch API und async/await getätigt werden?	6
Frage 6: Was sind die wichtigsten Status Codes rund um die REST-APIs?.....	8
Frage 7: Was bedeutet Authentifizierung und Autorisierung rund um REST-APIs?	9
Aufgabe 1b: HTTP-Verben Experimentieren.....	11
1: Abruf einer Liste (Get-Verb)	11
2: Abruf eines einzelnen Datensatzes	12
3: Löschen eines Datensatzes.....	12

Aufgabe 1a: Recherche zu REST APIs

Frage 1: Was ist eine REST API?

Eine REST-API (Representational State Transfer Application Programming Interface) ist eine Schnittstelle, die es verschiedenen Softwareanwendungen ermöglicht, miteinander zu kommunizieren und Daten auszutauschen. Sie basiert auf den Prinzipien von REST, einer Architekturstilrichtlinie für Netzwerkanwendungen.

RESTful APIs verwenden HTTP-Methoden (wie GET, POST, PUT, DELETE) und arbeiten in der Regel mit Ressourcen, die über URLs (Uniform Resource Locators) identifiziert werden. Die Daten werden oft im JSON- oder XML-Format übertragen, obwohl JSON heutzutage am häufigsten verwendet wird.

Ein RESTful API-Endpunkt repräsentiert eine bestimmte Ressource (z.B. ein Benutzerkonto, ein Produkt in einem Online-Shop) und ermöglicht es, Operationen auf dieser Ressource durchzuführen, wie das Abrufen, Erstellen, Aktualisieren oder Löschen von Daten.

RESTful APIs sind sehr flexibel und skalierbar, was sie zu einer beliebten Wahl für die Entwicklung von Webanwendungen und Services macht. Sie ermöglichen es, Anwendungen und Systeme zu integrieren und Daten zwischen ihnen auszutauschen, was in der heutigen vernetzten Welt von großer Bedeutung ist.

Frage 2: Was bewirken die unterschiedlichen HTTP-Verben für eine REST API?

Die verschiedenen HTTP-Verben (auch als Methoden bezeichnet) in einer RESTful API haben spezifische Bedeutungen und bewirken unterschiedliche Aktionen auf den Ressourcen, die durch die API dargestellt werden. Hier sind die häufigsten HTTP-Verben und ihre typischen Auswirkungen:

1. ****GET****: Der GET-Befehl wird verwendet, um eine Ressource abzurufen. Es wird verwendet, um Daten von der API anzufordern, ohne dabei die Ressource zu ändern. Es ist eine sichere Methode, da sie nur Leseprozesse ausführt und keine Änderungen an den Daten vornimmt.
2. ****POST****: Mit POST können neue Daten erstellt werden. Es wird verwendet, um eine neue Ressource zu erstellen oder Daten an die API zu senden. POST-Anfragen führen oft zu Änderungen am Serverzustand, wie beispielsweise das Hinzufügen eines neuen Datensatzes zu einer Datenbank.

3. **PUT**: PUT wird verwendet, um eine bestehende Ressource zu aktualisieren. Es ermöglicht es, eine Ressource durch eine neue Version zu ersetzen. Wenn eine Ressource unter der angegebenen URL bereits existiert, wird sie durch die neuen Daten ersetzt.

4. **PATCH**: Ähnlich wie PUT wird PATCH verwendet, um eine bestehende Ressource zu aktualisieren. Der Unterschied besteht darin, dass PATCH normalerweise verwendet wird, um Teile einer Ressource zu ändern, anstatt die gesamte Ressource zu ersetzen. PATCH ist nützlich, wenn Sie nur bestimmte Teile einer Ressource aktualisieren möchten.

5. **DELETE**: DELETE wird verwendet, um eine Ressource zu löschen. Es bewirkt, dass die angegebene Ressource vom Server entfernt wird.

Es ist wichtig zu beachten, dass RESTful APIs das Prinzip der Zustandslosigkeit befolgen, was bedeutet, dass jede Anfrage an den Server alle notwendigen Informationen enthalten muss, um die Anfrage zu verstehen. Der Server sollte den Zustand der Anwendung ausschließlich anhand der Anfrage selbst interpretieren können, ohne auf vergangene Anfragen zurückgreifen zu müssen.

Durch die Verwendung dieser verschiedenen HTTP-Verben können RESTful APIs eine klare und einheitliche Schnittstelle bieten, die es ermöglicht, auf Ressourcen zuzugreifen und mit ihnen zu interagieren.

[Frage 3: Gibt es weitere HTTP-Verben? Wenn ja, beschreibe sie ausführlich.](#)

Ja, neben den häufig verwendeten HTTP-Verben GET, POST, PUT, PATCH und DELETE gibt es auch einige weniger gebräuchliche Verben, die in bestimmten Kontexten verwendet werden können. Hier sind einige von ihnen:

1. **OPTIONS**: Die OPTIONS-Methode wird verwendet, um die unterstützten Methoden und Eigenschaften einer Ressource zu ermitteln, ohne die Ressource selbst zu ändern. Ein Server kann in der Antwort auf eine OPTIONS-Anfrage Informationen über erlaubte Methoden, unterstützte Authentifizierungsschemata und andere Informationen bereitstellen.

2. **HEAD**: HEAD ist ähnlich wie GET, aber es fordert nur die Header-Informationen einer Ressource an, ohne den eigentlichen Inhalt zurückzugeben. HEAD wird oft verwendet, um zu überprüfen, ob eine Ressource vorhanden ist und ob sie sich geändert hat, ohne den gesamten Inhalt herunterzuladen.

3. **TRACE**: TRACE wird verwendet, um eine Loopback-Diagnose einer Anfrage zu ermöglichen. Der Server, auf den die Anfrage gerichtet ist, antwortet mit einem Echo der empfangenen Anfrage. Dies kann verwendet werden, um festzustellen, ob eine Anfrage auf ihrem Weg zum Server verändert wurde.

4. **CONNECT**: CONNECT wird normalerweise für HTTP-Tunneling verwendet. Es ermöglicht es, eine Verbindung zu einem Proxy-Server herzustellen, der dann als Vermittler zwischen dem Client und einem anderen Server fungiert, beispielsweise für verschlüsselte Verbindungen.

Es ist wichtig zu beachten, dass einige dieser weniger gebräuchlichen Verben spezifische Anwendungsfälle und Sicherheitsüberlegungen haben. Sie werden nicht so häufig verwendet wie die grundlegenden Verben GET, POST, PUT, PATCH und DELETE, aber sie können in speziellen Situationen nützlich sein.

Wenn Sie eine RESTful API entwickeln oder verwenden, ist es wichtig, die am besten geeigneten Verben für die jeweiligen Aktionen zu wählen, um eine klare und konsistente API-Schnittstelle zu schaffen.

Frage 4: Wie sehen typische REST-Anfragen und Antworten aus?

Typische REST-Anfragen und -Antworten in einer RESTful API verwenden HTTP-Methoden und sind so strukturiert, dass sie auf Ressourcen in der API zugreifen und mit ihnen interagieren. Hier sind Beispiele für typische REST-Anfragen und entsprechende Antworten:

1. **GET-Anfrage**:

Anfrage (GET) zum Abrufen von Informationen über einen Benutzer mit der ID 123:

GET /api/users/123

Antwort (200 OK) mit den Benutzerdaten im JSON-Format:

```
{  
  "id": 123,  
  "name": "Max Mustermann",  
  "email": "max@example.com"  
}
```

2. ****POST-Anfrage****:

Anfrage (POST) zum Erstellen eines neuen Benutzers mit den bereitgestellten Daten:

POST /api/users

Anfragetextkörper (JSON):

```
{  
  "name": "Neuer Benutzer",  
  "email": "neu@example.com"  
}
```

Antwort (201 Created) mit Bestätigung und den erstellten Benutzerdaten:

```
{  
  "id": 124,  
  "name": "Neuer Benutzer",  
  "email": "neu@example.com"  
}
```

3. ****PUT-Anfrage****:

Anfrage (PUT) zur Aktualisierung der Benutzerdaten für Benutzer 123:

PUT /api/users/123

Anfragetextkörper (JSON) mit den aktualisierten Daten:

```
{
```

```
"name": "Geänderte Informationen",  
"email": "geaendert@example.com"  
}
```

Antwort (200 OK) zur Bestätigung der Aktualisierung:

```
{  
  "message": "Benutzerdaten wurden erfolgreich aktualisiert."  
}
```

4. ****DELETE-Anfrage****:

Anfrage (DELETE) zur Löschung eines Benutzers mit der ID 123:

```
DELETE /api/users/123
```

Antwort (204 No Content) zur Bestätigung der Löschung:

Es wird keine Antwort im Body geliefert.

Diese Beispiele zeigen typische Anfragen und Antworten in einer RESTful API. Die Anfragen verwenden HTTP-Methoden, um auf bestimmte Ressourcen (in diesem Fall Benutzer) zuzugreifen und mit ihnen zu interagieren. Die Antworten enthalten oft JSON- oder XML-Daten, die die Ergebnisse der Aktionen oder Informationen über die Ressourcen darstellen. Die HTTP-Statuscodes in den Antworten geben an, ob die Anfragen erfolgreich waren oder Fehler aufgetreten sind.

Frage 5: Wie können REST-Anfragen in JavaScript mit der Fetch API und async/await getätigt werden?

In JavaScript können REST-Anfragen mithilfe der `Fetch API` und der `async/await`-Syntax gemacht werden. Die `Fetch API` ermöglicht es, Ressourcen (wie JSON-Daten) von einem

Server zu holen. ``async/await`` ist eine moderne JavaScript-Funktion, die es erleichtert, asynchrone Operationen zu handhaben.

Hier ist ein Beispiel, wie man eine GET-Anfrage in JavaScript mit der ``Fetch API`` und ``async/await`` tätigt:

```
async function fetchUserData() {  
  try {  
    const response = await fetch('https://example.com/api/users/123');  
    const userData = await response.json();  
    return userData;  
  } catch (error) {  
    console.error('Fehler beim Abrufen der Benutzerdaten:', error);  
  }  
}
```

```
// Verwendung der Funktion  
fetchUserData().then(data => {  
  console.log('Benutzerdaten:', data);  
});  
...
```

In diesem Beispiel:

1. ``async function fetchUserData()`` definiert eine `async`-Funktion, die die Benutzerdaten abrufen.
2. ``await fetch('https://example.com/api/users/123')`` ruft die Ressource unter der angegebenen URL ab. Die ``await``-Anweisung pausiert die Ausführung, bis die Anfrage abgeschlossen ist und eine Antwort zurückgibt.

3. ``const userData = await response.json()`` liest die Antwort als JSON und gibt ein JavaScript-Objekt zurück.

4. Der Fehler wird in einem ``catch``-Block behandelt, falls bei der Anfrage ein Fehler auftritt.

5. Die Funktion ``fetchUserData()`` gibt ein Promise zurück, das die Benutzerdaten enthält.

6. Die Funktion wird dann mit ``then()`` aufgerufen, um die Benutzerdaten zu verwenden, wenn sie verfügbar sind.

Wenn Sie POST-, PUT- oder DELETE-Anfragen durchführen möchten, ändern Sie einfach die Methode im ``fetch``-Aufruf und fügen Sie den Anfragetextkörper hinzu, wenn nötig.

Beachten Sie, dass es wichtig ist, Fehler zu behandeln, um sicherzustellen, dass Ihre Anwendung robust ist und gut auf unerwartete Situationen reagieren kann.

Frage 6: Was sind die wichtigsten Status Codes rund um die REST-APIS?

HTTP-Statuscodes sind wichtige Rückgabewerte in einer RESTful API, die Informationen über den Erfolg oder das Scheitern einer HTTP-Anfrage liefern. Hier sind einige der wichtigsten HTTP-Statuscodes, die in Zusammenhang mit REST-APIs verwendet werden:

1. ****200 OK****: Die Anfrage wurde erfolgreich bearbeitet, und die Antwort enthält die angeforderten Daten.

2. ****201 Created****: Die Anfrage wurde erfolgreich bearbeitet, und als Ergebnis wurde eine neue Ressource erstellt. Die URL zur neu erstellten Ressource wird oft in der Antwort zurückgegeben.

3. ****204 No Content****: Die Anfrage wurde erfolgreich bearbeitet, und die Antwort enthält keine Daten. Dies wird häufig für erfolgreiche DELETE-Anfragen verwendet.

4. ****400 Bad Request****: Die Anfrage war fehlerhaft oder ungültig. Dies kann auf ungültige Parameter oder fehlende Pflichtfelder hinweisen.

5. ****401 Unauthorized****: Die Anfrage erfordert eine Authentifizierung, und die bereitgestellten Anmeldeinformationen sind ungültig oder fehlen.
6. ****403 Forbidden****: Der Server verweigert die Ausführung der Anfrage, selbst wenn die Authentifizierung erfolgreich war. Dies kann auf Berechtigungsprobleme hinweisen.
7. ****404 Not Found****: Die angeforderte Ressource wurde nicht gefunden.
8. ****405 Method Not Allowed****: Die Methode, die in der Anfrage verwendet wurde (GET, POST, PUT usw.), ist für die angeforderte Ressource nicht zulässig.
9. ****500 Internal Server Error****: Ein allgemeiner Serverfehler ist aufgetreten. Dieser Statuscode weist auf ein Problem auf Serverseite hin.
10. ****503 Service Unavailable****: Der Server ist vorübergehend nicht verfügbar. Dies kann auf Wartungsarbeiten oder Überlastung hinweisen.

Diese sind nur einige der häufigsten HTTP-Statuscodes. Es gibt viele andere Statuscodes, die spezifische Informationen über den Zustand der Anfrage und der Ressource liefern können. Es ist wichtig, die Statuscodes sinnvoll zu behandeln und angemessene Fehlermeldungen und Rückgabewerte in Ihrer API zu verwenden, um Client-Anwendungen über den Status der Anfrage zu informieren.

Frage 7: Was bedeutet Authentifizierung und Autorisierung rund um REST-APIS?

Authentifizierung und Autorisierung sind zwei wichtige Sicherheitsaspekte in Zusammenhang mit RESTful APIs, die sicherstellen, dass nur autorisierte Benutzer auf bestimmte Ressourcen und Aktionen zugreifen können.

****Authentifizierung**** bezieht sich auf den Prozess, bei dem ein Benutzer oder eine Anwendung ihre Identität nachweist. Dies geschieht in der Regel durch Bereitstellung von Anmeldeinformationen, wie Benutzername und Passwort, API-Token oder Zertifikate. Die Authentifizierung ermöglicht der API, den Benutzer zu identifizieren und sicherzustellen, dass sie die erforderlichen Berechtigungen für den Zugriff auf die Ressourcen haben.

Hier sind einige gängige Methoden der Authentifizierung in RESTful APIs:

1. ****Basic Authentication****: Der Benutzer sendet Benutzername und Passwort im Authorization-Header der Anfrage. Dies ist einfach zu implementieren, aber nicht besonders sicher, es sei denn, es wird über HTTPS verschlüsselt.
2. ****Token-Based Authentication****: Benutzer erhalten ein eindeutiges Token nach der Anmeldung, das sie für zukünftige Anfragen verwenden. Dieses Token wird oft in einem Header (z. B. "Authorization: Bearer <Token>") übertragen.
3. ****OAuth****: OAuth ist ein komplexeres Protokoll für die Authentifizierung, das häufig für die Autorisierung von Drittanbieteranwendungen verwendet wird.

****Autorisierung**** hingegen bezieht sich auf die Berechtigungen, die einem authentifizierten Benutzer gewährt werden. Sobald ein Benutzer authentifiziert ist, sollte die API prüfen, ob dieser Benutzer die erforderlichen Berechtigungen für die angeforderten Ressourcen oder Aktionen hat.

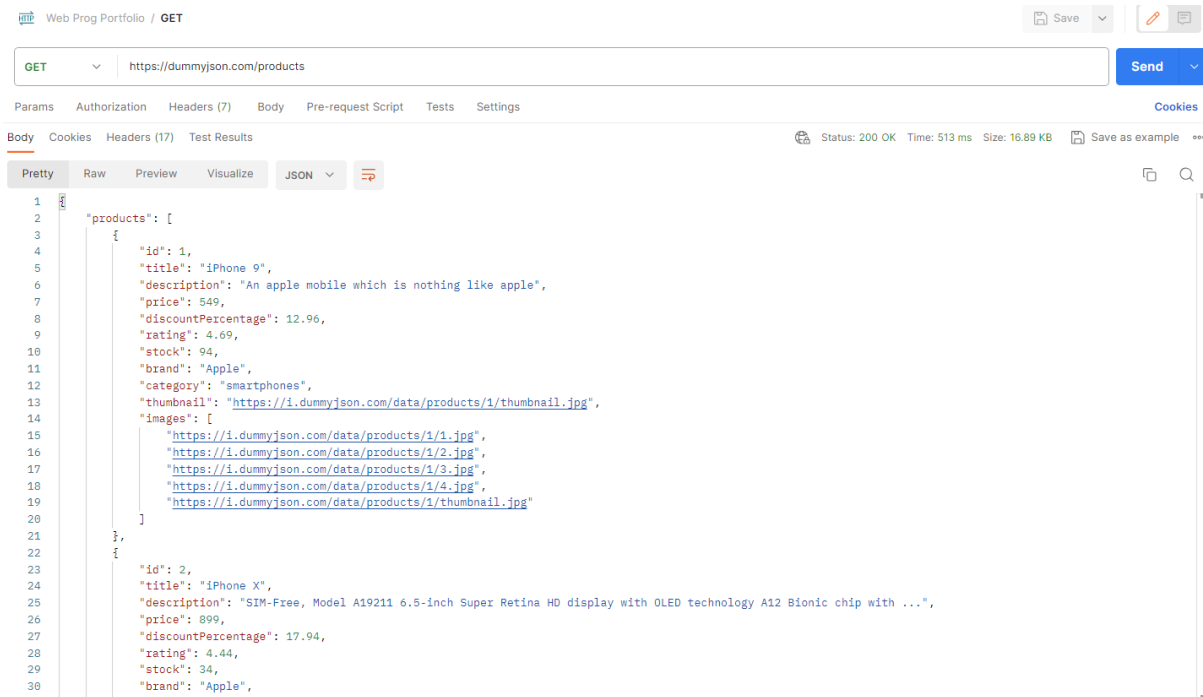
Hier sind einige gängige Ansätze zur Autorisierung in RESTful APIs:

1. ****Rollenbasierte Autorisierung****: Benutzer werden Rollen zugewiesen (z. B. "Benutzer", "Administrator", "Redakteur"), und der Zugriff auf Ressourcen wird basierend auf diesen Rollen gewährt.
2. ****Berechtigungsbasierte Autorisierung****: Benutzer erhalten bestimmte Berechtigungen (z. B. "Lesen", "Schreiben", "Löschen") für bestimmte Ressourcen, und der Zugriff wird basierend auf diesen Berechtigungen gesteuert.
3. ****Zugriffskontrolllisten (ACL)****: Mit ACLs können detaillierte Zugriffsberechtigungen auf Ressourcen festgelegt werden, wodurch eine fein abgestimmte Kontrolle ermöglicht wird.

Die Kombination von Authentifizierung und Autorisierung stellt sicher, dass nur authentifizierte Benutzer mit den richtigen Berechtigungen auf die API-Ressourcen zugreifen können. Dies ist entscheidend für die Sicherheit und den Datenschutz in RESTful APIs, insbesondere wenn vertrauliche oder geschützte Informationen verarbeitet werden. Die genauen Implementierungsdetails können je nach Anwendungsfall und Framework variieren.

Aufgabe 1b: HTTP-Verben Experimentieren

1: Abruf einer Liste (Get-Verb)



The screenshot shows a web browser interface with a REST client. The method is set to GET and the URL is <https://dummyjson.com/products>. The status is 200 OK, time is 513 ms, and size is 16.89 KB. The response body is displayed in JSON format, showing a list of two products.

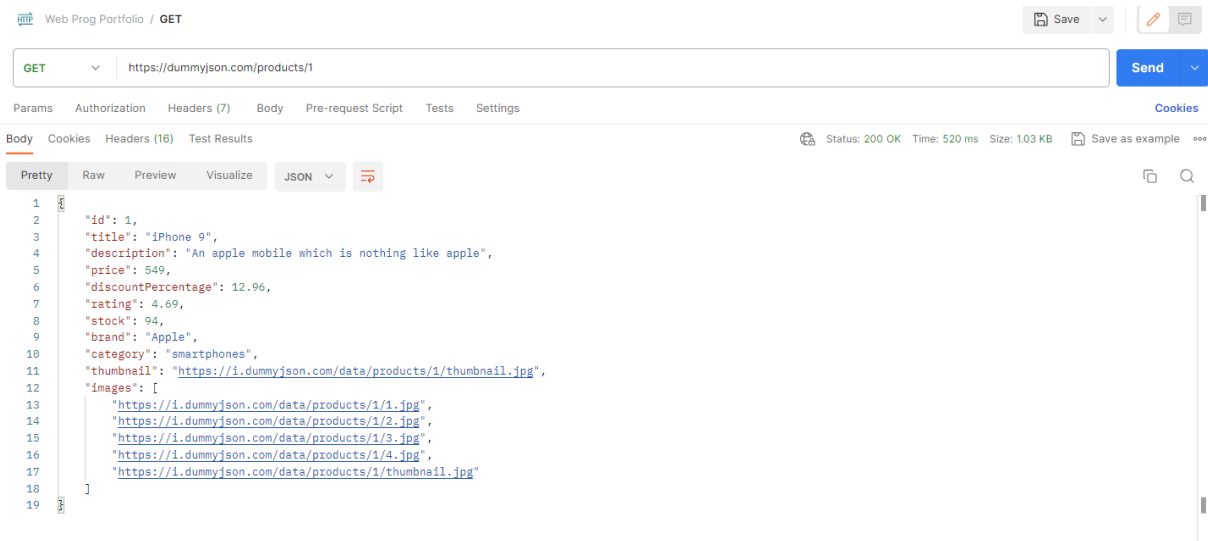
```
1  {
2    "products": [
3      {
4        "id": 1,
5        "title": "iPhone 9",
6        "description": "An apple mobile which is nothing like apple",
7        "price": 549,
8        "discountPercentage": 12.96,
9        "rating": 4.69,
10       "stock": 94,
11       "brand": "Apple",
12       "category": "smartphones",
13       "thumbnail": "https://i.dummyjson.com/data/products/1/thumbnail.jpg",
14       "images": [
15         "https://i.dummyjson.com/data/products/1/1.jpg",
16         "https://i.dummyjson.com/data/products/1/2.jpg",
17         "https://i.dummyjson.com/data/products/1/3.jpg",
18         "https://i.dummyjson.com/data/products/1/4.jpg",
19         "https://i.dummyjson.com/data/products/1/thumbnail.jpg"
20       ]
21     },
22     {
23       "id": 2,
24       "title": "iPhone X",
25       "description": "SIM-Free, Model A19211 6.5-inch Super Retina HD display with OLED technology A12 Bionic chip with ...",
26       "price": 899,
27       "discountPercentage": 17.94,
28       "rating": 4.44,
29       "stock": 34,
30       "brand": "Apple",
31       "category": "smartphones",
32       "thumbnail": "https://i.dummyjson.com/data/products/2/thumbnail.jpg",
33       "images": [
34         "https://i.dummyjson.com/data/products/2/1.jpg",
35         "https://i.dummyjson.com/data/products/2/2.jpg",
36         "https://i.dummyjson.com/data/products/2/3.jpg",
37         "https://i.dummyjson.com/data/products/2/thumbnail.jpg"
38       ]
39     }
40   ]
41 }
```

Der Statuscode(200) weist auf einen erfolgreichen Abruf hin. Unter Verwendung der GET-Methode wird signalisiert, dass eine Entität also Daten vom Server erwartet werden.

Die URL ist der Endpunkt und besteht aus Teilen: (<https://dummyjson.com/>) und der Pfad (products). Der Pfad enthält den Dateipfad zur Ressource und im allgemeinen kann der Server mithilfe des Pfads diejenige Ressourcen unter Verwendung des jeweiligen Verbes identifizieren und bereitstellen.

In diesem Fall wird ein Objekt bereitgestellt, welches eine Liste namens products als Antwort zurückgibt. Der Antwort erfolgt im JSON-Format.

2: Abruf eines einzelnen Datensatzes

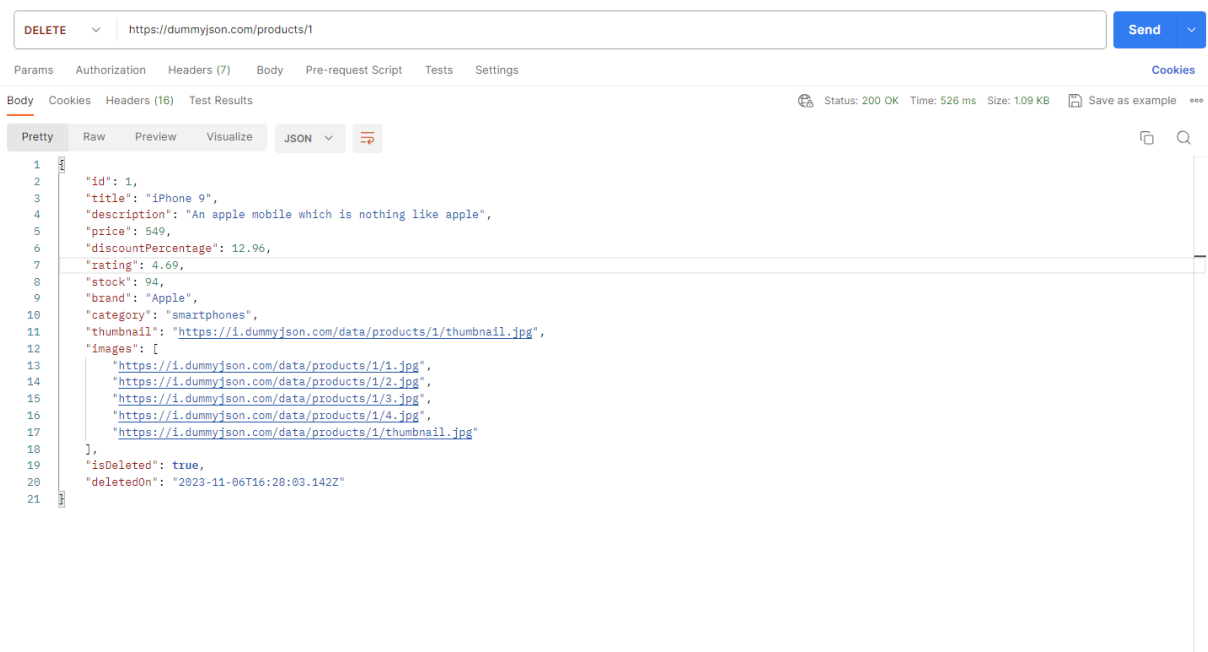


Auch in diesem Fall ist der Statuscode(200). Damit ist der Abruf unter Verwendung der GET-Methode wieder erfolgreich.

Auch in diesem Fall besteht Endpunkt wieder aus zwei Teilen: (<https://dummyjson.com/>) und der Pfad (products/1). Aber hier steht die Nummer 1 für den ID des Produktes und somit besteht unser Antwort nur aus einem Produkt mit dem ID 1.

Auch hier erfolgt der Antwort im JSON-Format.

3: Löschen eines Datensatzes



Auch in diesem Fall ist der Statuscode(200) was teilweise richtig ist. In vielen Fällen wird der Statuscode(204 No Content) angezeigt werden, weil der Datensatz gelöscht wurde und somit nichts anzuzeigen existiert.

In diesem Fall ist der Endpunkt gleich wie beim Abrufen eines einzelnen Datensatzes: (<https://dummyjson.com/>) und der Pfad (products/1). Aber hier wird die Delete-Methode verwendet und sie signalisiert das Löschen des Produktes mit dem ID 1.

Auch hier erfolgt der Antwort im JSON-Format. Ganz unten stehen noch zwei Variablen „isDeleted=true“ und „deletedOn=Datum“ Die signalisieren laut Dokumentation der dummyJSON API, dass der Datensatz gelöscht wurde.