Was ist eine REST API?

October 19, 2022

U Was ist eine REST API?

G Eine REST API (Representational State Transfer Application Programming Interface) ist eine Methode zur Kommunikation zwischen zwei Softwareanwendungen über das HTTP-Protokoll. Sie ist eine Art und Weise der Schnittstelle für Webanwendungen und ermöglicht es verschiedenen Systemen, miteinander zu interagieren, indem die HTTP-Anfragen senden und empfangen.

Hier sind einige wichtige Merkmale und Prinzipien einer REST API:

1. **Ressourcen:** In einer REST-API werden Ressourcen durch eindeutige URIs (Uniform Resource Identifiers) identifiziert. Ressourcen können Objekte, Daten oder Dienste sein, die über die API verfügbar gemacht werden.
2. **HTTP-Verben:** REST verwendet HTTP-Verben wie GET, POST, PUT und DELETE, um Aktionen auf Ressourcen auszuführen. Zum Beispiel wird GET verwendet, um Daten abzurufen, POST wird verwendet, neue Ressourcen, PUT zum Aktualisieren von Ressourcen und DELETE zum Löschen von Ressourcen.
3. **Darstellung:** Ressourcen werden in verschiedenen Formaten dargestellt, wie JSON oder XML. Die Client-Anwendung kann das gewünschte Format durch Verhandlung mit dem Server auswählen.
4. **Zustandslosigkeit:** RESTful APIs sind zustandslos, was bedeutet, dass jede Anfrage vom Client alle erforderlichen Informationen enthält, um sie zu verstehen. Der Server speichert keine Informationen über den Zustand des Clients zwischen Anfragen.
5. **Verknüpfung:** RESTful APIs können hyperlinks in den API-Antworten einbeziehen, um die Navigation zwischen Ressourcen zu erleichtern. Dies ermöglicht eine dynamische Entdeckung von Ressourcen.
6. **Client-Server-Architektur:** REST trennt den Client und den Server voneinander, sodass die Skalierbarkeit und Wartbarkeit des Systems verbessert werden.

REST ist ein weit verbreitetes Konzept in der Webentwicklung und wird oft in Verbindung mit Webanwendungen, Mikrodiensten und der Integration von Anwendungen verwendet. Es bietet eine einfache und effektive Möglichkeit für Anwendungen, miteinander zu kommunizieren und auf Ressourcen zuzugreifen.

U Was bewirken die unterschiedlichen HTTP-Verben für eine REST API?

G Die verschiedenen HTTP-Verben in einer REST-API werden zur Unterscheidung verschiedener Arten von Operationen auf den Ressourcen, die von der API verwaltet werden. Hier sind die gängigsten HTTP-Verben und ihre Bedeutungen in einer REST-API:

1. **GET:** Dies dient dem Abrufen von Daten aus einer Ressource abzurufen. Wenn ein Client eine GET-Anfrage an eine bestimmte URI sendet, ruft der Server mit der angefragten Daten, ohne den Zustand der Ressource zu ändern. GET-Anfragen sollten idempotent sein, das bedeutet, dass sie keine Änderungen an der Ressource verursachen sollten.
2. **POST:** Das POST-Verb wird verwendet, um eine neue Ressource zu erstellen. Der Client sendet Daten an den Server, und der Server erstellt dann eine neue Ressource auf Grundlage dieser Daten. POST-Anfragen können nicht idempotent sein, da sie bei wiederholter Ausführung verschiedene Ressourcen erstellen können.
3. **PUT:** Das PUT-Verb wird verwendet, um eine vorhandene Ressource zu aktualisieren oder zu erstellen. Der Client sendet die gesamten Daten der Ressource an den Server, und der Server aktualisiert die Ressource gemäß den angegebenen Daten. PUT-Anfragen sollten idempotent sein, da mehrere aufeinanderfolgende PUT-Anfragen mit denselben Daten den gleichen Zustand der Ressource erzeugen sollten.
4. **PATCH:** Das PATCH-Verb ähnelt PUT, wird jedoch verwendet, um eine Teilaktualisierung einer Ressource durchzuführen. Der Client sendet nur die geänderten Teile der Ressource an den Server, anstatt die gesamte Ressource zu ersetzen. Dies ist nützlich, wenn Sie nur bestimmte Felder einer Ressource ändern möchten.
5. **DELETE:** Dies DELETE-Verb wird verwendet, um eine Ressource zu löschen. Wenn der Client einen DELETE-Anfrage an eine bestimmte URI sendet, entfernt der Server die entsprechende Ressource dauerhaft.

Zusätzlich zu diesen gängigen HTTP-Verben können RESTful APIs auch benutzerdefinierte Verben und Aktionen unterstützen, indem sie zusätzliche Pfade oder URI-Parameter verwenden. Die Verwendung der richtigen HTTP-Verben in einer REST-API ist wichtig, um die Semantik und die Erwartungen der Interaktionen mit den Ressourcen klar zu definieren und um sicherzustellen, dass die API gut geplant und verständlich ist.

U Wie sehen typische REST-Anfragen und Antworten aus?

G Typische REST-Anfragen und -Antworten verwenden HTTP-Verben, URIs, Header und Datenformate, um mit Ressourcen zu interagieren. Hier sind Beispiele für typische REST-Anfragen und -Antworten:

GET-Anfrage (Daten abrufen):

Anfrage:

http Copy code
```
GET /api/users/123 HTTP/1.1
Host: example.com
Accept: application/json
```

In diesem GET-Anfrage wird angefordert, die Benutzerressource mit der ID 123 abzurufen. Der "Accept"-Header gibt an, dass der Client JSON als Datenformat bevorzugt.

Antwort:

http Copy code
```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 123,
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

Die Antwort enthält den Statuscode "200 OK" und JSON-Daten, die die Informationen über den Benutzer mit ID 123 darstellen.

POST-Anfrage (Neue Ressource erstellen):

Anfrage:

http Copy code
```
POST /api/users HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "name": "Jane Smith",
  "email": "jane.smith@example.com"
}
```

In diesem POST-Anfrage sendet der Client JSON-Daten, um eine neue Benutzerressource zu erstellen.

Antwort:

http Copy code
```
HTTP/1.1 201 Created
Content-Type: application/json
Location: /api/users/124

{
  "id": 124,
  "name": "Jane Smith",
  "email": "jane.smith@example.com"
}
```

Die Antwort enthält den Statuscode "201 Created" und zeigt die erstellte Ressource an. Der "Location"-Header gibt die URI der neu erstellten Ressource an.

PUT-Anfrage (Ressource aktualisieren):

Anfrage:

http Copy code
```
PUT /api/users/123 HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "name": "John W. Doe"
}
```

In diesem PUT-Anfrage werden die Daten eines Benutzers (ID 123) aktualisiert.

Antwort:

http Copy code
```
HTTP/1.1 204 No Content
```

Die Antwort hat den Statuscode "204 No Content", was bedeutet, dass die Aktualisierung erfolgreich war, aber keine Daten im Antwortkörper zurückgegeben werden.

DELETE-Anfrage (Ressource löschen):

Anfrage:

http Copy code
```
DELETE /api/users/123 HTTP/1.1
Host: example.com
```

In diesem DELETE-Anfrage wird die Benutzerressource mit ID 123 gelöscht.

Antwort:

http Copy code
```
HTTP/1.1 204 No Content
```

Die Antwort hat den Statuscode "204 No Content", was bedeutet, dass die Ressource erfolgreich gelöscht wurde.

Diese Beispiele zeigen typische REST-Anfragen und -Antworten, die HTTP-Verben, URIs, Header und Datenformate verwenden, um auf Ressourcen zuzugreifen, sie zu erstellen, zu aktualisieren und zu löschen. Die genauen Details können je nach API und Implementierung variieren, aber die grundlegenden Prinzipien bleiben gleich.

U Wie können REST-Anfragen in JavaScript mit der Fetch API oder ähnlich asynchron gestaltet werden?

G In JavaScript können REST-Anfragen mithilfe der `fetch` API und `async/await` asynchron gestaltet werden. Die `fetch` API bietet eine moderne Möglichkeit, HTTP-Anfragen an Server zu senden und die Antwort zu verarbeiten. Der `async/await` Syntax ermöglicht es, asynchrone Anfragen auf eine prägnante und lesbare Weise zu verwalten. Hier ist ein Beispiel, wie REST-Anfragen in JavaScript mit der `fetch` API und `async/await` gestaltet werden:

javascript Copy code
```
async function fetchUserAndUpdate(userId) {
  const apiUrl = `https://api.example.com/users/${userId}`;

  try {
    const response = await fetch(apiUrl, {
      method: 'GET',
      headers: {
        'Accept': 'application/json',
        // Fügen Sie hier weitere Header hinzu, falls benötigt.
      },
    });

    if (!response.ok) {
      throw new Error('HTTP-Fehler: Status=${response.status}');
    }

    const userData = await response.json();
    console.log('Benutzerdaten:', userData);

    // Aktualisieren Sie die Benutzerdaten hier.
    const updatedUserData = { ...userData, name: 'Neuer Name' };
    const updateResponse = await fetch(apiUrl, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
        // Fügen Sie hier weitere Header hinzu, falls benötigt.
      },
      body: JSON.stringify(updatedUserData),
    });

    if (!updateResponse.ok) {
      throw new Error('HTTP-Fehler: Status=${updateResponse.status}');
    }

    const updatedUserResponseData = await updateResponse.json();
    console.log('Aktualisierte Benutzerdaten:', updatedUserResponseData);
  } catch (error) {
    console.error('Fehler:', error);
  }
}

// Beispielaufruf der Funktion
const userId = 123;
fetchUserAndUpdate(userId);
```

In diesem Beispiel wird die `fetchUserAndUpdate` Funktion erstellt, um Benutzerdaten von einer REST-API abzurufen, hier und die Antwort zu verarbeiten.

1. Die URL, die Ressource wird zusammengestellt, die Sie abrufen möchten.
2. Eine `fetch` Anfrage wird mit der URL und den erforderlichen Optionen erstellt, wobei `"GET"` als Methode verwendet wird. Sie können auch weitere Header hinzufügen, wenn erforderlich.
3. Die Anfrage wird mit `"await"` asynchron abgewartet, und die Antwort wird in der Variable `"response"` gespeichert.
4. Wenn die Anfrage erfolgreich war (Statuscode 200 bis 299), wird die Antwort in JSON umgewandelt und in der Variable `"userData"` gespeichert.
5. Wenn ein HTTP-Fehler auftritt, wird eine Ausnahme ausgelöst und in der `"catch"` Klausel behandelt.
6. In der Beispielaufruffunktion wird die `"fetchUserAndUpdate"` Funktion aufgerufen, und die Daten werden anschließend verarbeitet.

Denken Sie daran, dass dies ein einfaches Beispiel ist, und in einer echten Anwendung sollten Sie eine umfassendere Fehlerbehandlung und andere Anforderungen berücksichtigen, die Ihre API möglicherweise hat.

U Welche Best Practices gibt es für die Dokumentation von REST APIs?

G Die Dokumentation von RESTful APIs ist entscheidend, um Entwicklern das Verständnis und die Nutzung Ihrer API zu erleichtern. Hier sind einige bewährte Praktiken für die Dokumentation von REST-APIs:

1. **Klare und konsistente Namensgebung:** Verwenden Sie klare und konsistente Namen für Ressourcen, URIs, HTTP-Verben und Felder in den Anfragen und Antworten. Dies erleichtert die Verwendung und das Verständnis der API.
2. **Verwendbare URIs:** Entwerfen Sie sauberne und lesbare URIs, die die Hierarchie und Struktur Ihrer API widerspiegeln. Vermeiden Sie komplexe und unübersichtliche URIs.
3. **Beschreibende Dokumentation:** Bieten Sie eine klare und detaillierte Beschreibung der API und ihrer Endpunkte. Beschreiben Sie, welche Ressourcen verfügbar sind, wie sie genutzt werden und welche Parameter akzeptiert werden.
4. **Nutzung von HTTP-Statuscodes:** Verwenden Sie die richtigen HTTP-Statuscodes in Ihren Antworten, um den Status der Anfrage klar zu kommunizieren. 2. B. 200 für Erfolg, 404 für nicht gefunden, 400 für ungültige Anfrage usw.
5. **Datenformat:** Geben Sie an, welches Datenformat für Anfragen und Antworten verwendet wird. JSON ist heute sehr verbreitet, aber Sie sollten auch alternative Formate, wie XML, unterstützen, wenn dies erforderlich ist.
6. **Authentifizierung und Autorisierung:** Erklären Sie, wie die Authentifizierung und Autorisierung in Ihrer API funktioniert. Geben Sie an, welche Authentifizierungsmethoden (z. B. API-Token, OAuth) unterstützt werden.
7. **Beispielanfragen und -antworten:** Bieten Sie Beispiele für Anfragen und Antworten bereit, um Entwicklern zu zeigen, wie die API verwendet werden kann. Dies erleichtert das Verständnis und die Integration.
8. **Rate Limiting:** Falls Ihre API Rate Limiting oder Zugriffsbeschränkungen verwendet, aufzuzeichnen Sie dies deutlich und erklären Sie die Regeln und Limits.
9. **Hypermedia Links:** Wenn Ihre API Hypermedia Links verwendet, um die Navigation zwischen Ressourcen zu erleichtern, dokumentieren Sie diese Links und wie sie verwendet werden können.
10. **Versionskontrolle:** Wenn Sie Änderungen an Ihrer API vornehmen, sollten Sie eine klare Versionierung einführen und dokumentieren, wie Entwickler auf verschiedene API-Versionen zwischen ändern können.
11. **Tools und SDKs:** Bieten Sie Entwicklern Tools, SDKs (Software Development Kits) und Beispielcode in verschiedenen Programmiersprachen an, um die Integration Ihrer API zu erleichtern.
12. **Feedback Möglichkeiten:** Bieten Sie Möglichkeiten für Entwickler, Feedback zu geben, Fragen zu stellen oder auf Probleme hinzuweisen.
13. **Aktualisieren Sie die Dokumentation regelmäßig:** Eine API und die damit verbundene Dokumentation werden sich im Laufe der Zeit ändern. Stellen Sie sicher, dass Ihre Dokumentation regelmäßig aktualisiert wird, um mit den APIs Änderungen Schritt zu halten.

Eine gut dokumentierte API fördert die Akzeptanz und Nutzung durch Entwickler und trägt zur erfolgreichen Integration und Verwendung Ihrer Dienste bei. Die oben genannten Best Practices sind eine gute Grundlage für die Erstellung einer informativen und benutzerfreundlichen API-Dokumentation.

U Welche Alternativen zu einer Rest API gibt es für Entwickler und was sind deren Vorteile und Nachteile?

G Es gibt verschiedene Alternativen zu RESTful APIs, je nach den Anforderungen und Zielen eines Projekts. Hier sind einige alternative Ansätze und ihre jeweiligen Vor- und Nachteile:

1. **GraphQL:**
   - **Vorteile:** GraphQL ermöglicht es Clients, genau die Daten anzufordern, die sie benötigen, anstatt vordefinierte Datenstrukturen zu erhalten. Dies minimiert Overfetching (das Herunterladen von mehr Daten, als benötigt werden) und Underfetching (das Herunterladen von zu wenig Daten). Entwickler haben mehr Kontrolle über die abgerufenen Daten.
   - **Nachteile:** Die Implementierung von GraphQL kann komplexer sein bei REST, da Sie einen GraphQL-Server benötigen, um dynamische Anfragen zu verarbeiten. Die Lernkurve für Entwickler, die mit GraphQL nicht vertraut sind, kann steil sein.
2. **Websockets:**
   - **Vorteile:** Websockets ermöglichen eine bidirektionale Kommunikation zwischen Client und Server in Echtzeit. Dies ist nützlich für Echtzeit-Anwendungen, Chat-Anwendungen und Spiele, bei denen eine ständige Kommunikation erforderlich ist.
   - **Nachteile:** WebSockets erfordern eine dauerhafte Verbindung und können die Serverressourcen stärker belasten als REST-Anfragen. Sie sind nicht für alle Arten von Anwendungen geeignet.
3. **gRPC:**
   - **Vorteile:** gRPC ist ein von Google entwickeltes Protokoll, das effizientere, bidirektionale Kommunikation bietet HTTP/2 verwendet. Es verwendet Protocol Buffers (Datenformat, was zu effizienteren und kompakteren Nachrichten führt.
   - **Nachteile:** gRPC ist eine komplexere und weniger verbreitete Technologie als REST und Websockets. Es ist nicht so einfach zu implementieren.
4. **SOAP (Simple Object Access Protocol):**
   - **Vorteile:** SOAP ist ein älteres, aber immer noch verwendetes Protokoll für die Remote Procedure Call (RPC) Kommunikation zwischen Anwendungen. Es ist gut in Unternehmensumgebungen etabliert und bietet starke Typisierung und Sicherheit.
   - **Nachteile:** SOAP ist komplex und erzeugt oft umfangreiche XML-Payloads, was zu erhöhtem Overhead führen kann. Es wird oft als veraltet betrachtet im Vergleich zu REST und anderen modernen Ansätzen angesehen.
5. **JSON-RPC und XML-RPC:**
   - **Vorteile:** Diese Protokolle ermöglichen RPC-ähnliche Kommunikation über HTTP oder andere Protokolle. Sie sind einfacher und leichter in entwickelnden Umgebungen verwendet werden.
   - **Nachteile:** Sie können die gleichen Nachteile wie SOAP aufweisen, insbesondere hinsichtlich der Komplexität und der Datenübertragungsgröße.
6. **Server-Sent Events (SSE):**
   - **Vorteile:** SSE ermöglichen serverseitige Push-Benachrichtigungen an den Client, was für Echtzeit-Updates und Benachrichtigungen nützlich ist.
   - **Nachteile:** SSE ist auf unidirektionale Kommunikation beschränkt und erfordert eine ständige Verbindung zum Server zum Client. Es ist nicht für alle Arten von Anwendungen geeignet.

Die Wahl zwischen diesen Alternativen hängt von den Anforderungen Ihres Projekts, der Art der Anwendung und den Präferenzen der Entwickler ab. REST bleibt jedoch nach wie vor eine weit verbreitete und bewährte Methode zur Erstellung von Web-APIs aufgrund seiner Einfachheit und Verbreitung. Andere Ansätze können Vorteile in bestimmten Szenarien bieten und sollten sorgfältig und gründlich bewertet werden.

Continue this conversation

Terms of Service | Notice at collection | Privacy policy