REST API Design Standards - Do They Even Exist?

ADAM DUVANDER
NOVEMBER 26 2021

For a technology that is so ubiquitous, there is still confusion around REST standards. REST APIs are the primary form of web services, used widely to power websites, mobile applications, and most enterprise integrations. Compared to the previous generation of web services, namely SOAP and XML-RPC, there is much more flexibility with how companies create RESTful web services. This is because REST was developed to work in a large variety of environments, with multiple data types. Implementing a REST API is versatile, but there are still best practices and guidelines to consider.

In this post, we'll cover the REST API architectural style (REST itself is not a standard), some REST API design and naming conventions, and introduce a standard related to REST that can bring a standard-like rigor to your APIs. REST API design patterns do exist, and you can benefit by adopting them.

REST is an Architecture, Not a Standard

REST, which stands for REpresentational State Transfer, was introduced by Roy Fielding in his 2000 dissertation. While Fielding described REST outside of HTTP, it was developed alongside the protocol and is most commonly used over HTTP. While HTTP is a standard, REST itself is not. Rather, it's an architectural style that provides constraints that guide REST API design.

Many APIs do not conform to every element of REST, which has caused some to use the term RESTful to describe the most common types of APIs. Commonly-adopted principles of REST APIs include:

- **Client-server separation**: the client determines how responses are displayed to the user, allowing the server to parse the request and produce the response.
- **Stateless requests**: the server does not have to store any context between requests—everything needed is within each request.
- **Resource identifiers**: the interface is designed around resources that are identified by URLs.

These design patterns are present in most modern REST APIs. You can read more details in **Fielding's dissertation**, but we'll focus here on practical applications of REST in API design.

REST API Design Guidelines and Conventions

While REST is not a standard, there are guidelines and conventions that have been widely adopted. These include building atop HTTP's standards, naming resources as nouns, and incorporating popular data formats.

HTTP Methods and Examples

Web developers are likely familiar with GET and POST, along with the other HTTP methods, also sometimes called HTTP verbs. These methods define the type of request being made to a REST API.

Common HTTP methods include:

- **GET**: retrieve data, the equivalent to a **read in CRUD APIs**
- **POST**: add *new* data
- **PUT**: update existing data
- **PATCH**: update a *subset* of existing data
- **DELETE**: remove data

While there are other methods, they are rarely seen in REST APIs. The method itself does not mean much without the target resource.

Resource Names

Resources are sometimes referred to as the nouns that the HTTP verbs act upon. Earlier web services were built around remote procedure calls, which saw APIs as extensions of the code that called them. By contrast, REST resources can be accessed with multiple HTTP methods.

For example, if your API stored animals, a resource name might be “animals.” The path to access that resource might be `/api/animals`. The resource would then be combined with HTTP methods like so:

- `GET /api/animals` : retrieve a list of animals
- `POST /api/animals` : add a new animal

- `GET /api/animals/dog` : retrieve a single animal by ID
- `PUT /api/animals/dog` : update a single animal by ID
- `DELETE /api/animals/dog` : delete an animal by ID

Los identificadores pueden ser números enteros, valores hash u otros valores que se generan automáticamente. Si los recursos son plurales o singulares es una cuestión de preferencia (y muy debatida). Lo que es más importante, mantenga la coherencia dentro de cada API y entre las API de la misma organización.

Formatos de datos

La mayoría de las solicitudes de API devolverán contenido del servidor que el cliente necesita interpretar. En raras ocasiones, este contenido es texto sin formato; por lo general, utilizará un formato de datos estructurados. Si bien REST no especifica ningún formato de datos, JSON y XML son los dos más utilizados.

Un solo resultado de los estándares de la API JSON podría tener este aspecto:

```
{
  "id": "dog",
  "name": "Pet dog",
  "genus": "Canis",
  "img":
"https://cdn2.thedogapi.com/images/1MZ0YbOpS.jpg
"
}
```

Mientras que los mismos datos en XML podrían representarse así:

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <id>dog</id>
  <name>Domestic dog</name>
  <genus>Canis</genus>
  <img>https://cdn2.thedogapi.com/images/1MZ0YbOpS.jpg
</img>
</root>
```

Alternativamente, en RSS podría representarse así:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
<title> dog</title>
<link>"https://cdn2.thedogapi.com/images/1MZ0YbOpS.jpg"
</link>
<description>Pet dog</description>
</channel>
```

Se usarían estructuras similares en el cuerpo de las solicitudes al pasar datos, como con las solicitudes POST, PUT o PATCH. Normalmente, los cuerpos de solicitud y respuesta utilizan el mismo formato de datos.

Otros formatos comunes incluyen: CSV, HTML y YAML.

Estados HTTP

Dado que las API de REST dependen de los estándares HTTP, el estado de cada solicitud se utiliza para comunicar el resultado de la solicitud, como éxito o fracaso. Cada código de estado proporciona una respuesta legible por máquina, además de un mensaje legible por humanos. Los desarrolladores web (y varios usuarios) estarán familiarizados con muchos de estos.

- **200** : Éxito
- **201** : Creado
- **401** : no autorizado
- **403** : Prohibido
- **404** : No encontrado
- **429** : Demasiadas solicitudes

Hay más, por supuesto, incluida la redirección de nivel 300 y los errores del servidor de nivel 500. Querrá usar el código de estado correcto para la situación adecuada a medida que diseñe su API REST.

OpenAPI le permite establecer su propio "estándar"

Si bien REST no es un estándar, existen muchos otros estándares a menudo asociados con REST. Por ejemplo, OAuth cubre la autorización de recursos de terceros, mientras que JSON PATCH describe un enfoque estándar del método HTTP PATCH para el formato de datos JSON. Un estándar importante a tener en cuenta al diseñar sus propias API es la especificación OpenAPI.

OpenAPI es un estándar para describir las API REST y le permite declarar su método de seguridad API, puntos finales de diseño, datos de solicitud/respuesta y mensajes de estado HTTP. Juntos, estos definen su API en un solo documento. Estos son útiles durante la fase de diseño, pero también pueden ser útiles a lo largo del ciclo de vida de la API.

Cree documentos OpenAPI usando un editor visual, o incorpore un repositorio existente y use la interfaz gráfica para hacer cambios. Los estándares de diseño

de API REST existen para ayudar a todos los desarrolladores web a crear API mejores, más funcionales y claramente comunicadas. Ayude a crear definiciones del estándar API REST para su organización y cree API consistentes y fáciles de usar para desarrolladores **definiendo las suyas propias** , u obtenga más información sobre **los patrones de diseño para las API REST.**

Artículos Relacionados

JULIA SEIDMAN | 18 DE OCTUBRE DE 2022

Diseñar Seguridad con Estilo (Guías)

Pruebe la plantilla de guía de estilo del conjunto de reglas de seguridad de OWASP disponible en Stoplight Platform o en nuestra herramienta de linting de código abierto,...

PARÍN SHAH | 11 DE OCTUBRE DE 2022

Configuración de un equipo de diseño de Killer API (Parte 2)

¡Reúna estratégicamente el equipo de sus sueños con Stoplight Teams! Disponible para los planes Pro y Enterprise, Workspace Teams hace...