Code Your First API With Node.js and Express: Understanding REST APIs

Tania Rascia Aug 21, 2018 (Updated Aug 27, 2022)

47 likes | Read Time: 10 mins | English

Node.js REST API JavaScript Express Back-End

This post is part of a series called **Code Your First API With Node.js and Express**.

Code Your First API With Node.js and Express: Set Up the Server

Understanding REST and RESTful APIs

If you've spent any amount of time with modern web development, you will have come across terms like REST and API. If you've heard of these terms or work with APIs but don't have a complete understanding of how they work or how to build your own API, this series is for you.

In this tutorial series, we will start with an overview of REST principles and concepts. Then we will go on to create our own full-fledged API that runs on a Node.js Express server and connects to a MySQL database. After finishing this series, you should feel confident building your own API or delving into the documentation of an existing API.

Prerequisites

In order to get the most out of this tutorial, you should already have some basic command line knowledge, know the fundamentals of JavaScript, and have Node.js installed globally.

What Are REST and RESTful APIs?

Representational State Transfer, or **REST**, describes an architectural style for web services. REST consists of a set of standards or constraints for sharing data between different systems, and systems that implement REST are known as RESTful. REST is an abstract concept, not a language, framework, or type of software.

A loose analogy for REST would be keeping a collection of vinyl vs. using a streaming music service. With the physical vinyl collection, each record must be duplicated in its entirety to share and distribute copies. With a streaming service, however, the same music can be shared in perpetuity via a reference to some data such as a song title. In this case, the streaming music is a RESTful service, and the vinyl collection is a non-RESTful service.

An **API** is an Application Programming Interface, which is an interface that allows software programs to communicate with each other. A **RESTful API** is simply an API that adheres to the principles and constraints of REST. In a Web

API, a server receives a **request** through a URL endpoint and sends a **response** in return, which is often data in a format such as JSON.

REST Principles

Six guiding constraints define the REST architecture, outlined below.

1. **Uniform Interface**: The interface of components must be the same. This means using the URI standard to identify resources—in other words, paths that could be entered into the browser's location bar.
2. **Client-Server**: There is a separation of concerns between the server, which stores and manipulates data, and the client, which requests and displays the response.
3. **Stateless Interactions**: All information about each request is contained in each individual request and does not depend on session state.
4. **Cacheable**: The client and server can cache resources.
5. **Layered System**: The client can be connected to the end server, or an intermediate layer such as a load-balancer.
6. **Code on Demand (Optional)**: A client can download code, which reduces visibility from the outside.

Request and Response

You will already be familiar with the fact that all websites have URLs that begin with `http` (or `https` for the secure version). HyperText Transfer Protocol, or **HTTP**, is the method of communication between clients and servers on the internet.

We see it most obviously in the URL bar of our browsers, but HTTP can be used for more than just requesting websites from servers. When you go to a URL on the web, you are actually doing a `GET` request on that specific resource, and the website you see is the body of the response. We will go over `GET` and other types of requests shortly.

HTTP works by opening a **TCP** (Transmission Control Protocol) connection to a server port (`80` for `http`, `443` for `https`) to make a request, and the listening server responds with a status and a body.

A request must consist of a URL, a method, header information, and a body.

Request Methods

There are four major HTTP methods, also referred to as HTTP verbs, that are commonly used to interact with web APIs. These methods define the action that will be performed with any given resource.

HTTP request methods loosely correspond to the paradigm of **CRUD**, which stands for *Create, Update, Read, Delete*. Although CRUD refers to functions used in database operations, we can apply those design principles to HTTP verbs in a RESTful API.

Action	Request Method	Definition
Read	`GET`	Retrieves a resource
Create	`POST`	Creates a new resource

Create	`POST`	Creates a new resource
Update	`PUT`	Updates an existing resource
Delete	`DELETE`	Deletes a resource

`GET` is a safe, read-only operation that will not alter the state of a server. Every time you hit a URL in your browser, such as `https://www.google.com`, you are sending a `GET` request to Google's servers.

`POST` is used to create a new resource. A familiar example of `POST` is signing up as a user on a website or app. After submitting the form, a `POST` request with the user data might be sent to the server, which will then write that information into a database.

`PUT` updates an existing resource, which might be used to edit the settings of an existing user. Unlike `POST`, `PUT` is **idempotent**, meaning the same call can be made multiple times without producing a different result. For example, if you sent the same `POST` request to create a new user in a database multiple times, it would create a new user with the same data for each request you made. However, using the same `PUT` request on the same user would continuously produce the same result.

`DELETE`, as the name suggests, will simply delete an existing resource.

Response Codes

Once a request goes through from the client to the server, the server will send back an HTTP response, which will include metadata about the response known as headers, as well as the body. The first and most important part of the response is the **status code**, which indicates if a request was successful, if there was an error, or if another action must be taken.

The most well-known response code you will be familiar with is `404`, which means `Not Found`. `404` is part of the `4xx` class of status codes, which indicate client errors. There are five classes of status codes that each contain a range of responses.

- `1xx`: Information
- `2xx`: Success
- `3xx`: Redirection
- `4xx`: Client Error
- `5xx`: Server Error

Other common responses you may be familiar with are `301 Moved Permanently`, which is used to redirect websites to new URLs, and `500 Internal Server Error`, which is an error that comes up frequently when something unexpected has happened on a server that makes it impossible to fulfil the intended request.

With regards to RESTful APIs and their corresponding HTTP verbs, all the responses should be in the `2xx` range.

Request	**Response**
`GET`	`200` OK
`POST`	`201` Created
`PUT`	`200` OK
`DELETE`	`200` OK, `202` Accepted, or `204` No Content

`200 OK` is the response that indicates that a request is successful. It is used as a response when sending a `GET` or `PUT` request. `POST` will return a `201 Created` to indicate that a new resource has been created, and `DELETE` has a few acceptable responses, which convey that either the request has been accepted (`202`), or there is no content to return because the resource no longer exists (`204`).

If you do not get a request in the 2xx range, that could mean many things. 1xx requests send extra information, 3xx requests mean redirects, 4xx requests mean client errors, and 5xx requests mean server errors.

We can test the status code of a resource request using cURL, which is a command-line tool used for transferring data via URLs. Using `curl`, followed by the `-i` or `--include` flag, will send a `GET` request to a URL and display the headers and body. We can test this by opening the command-line program and testing cURL with Google.

```
curl -i https://www.google.com
```

Google's server will respond with the following.

```
HTTP/2 200
date: Sun, 21 Aug 2022 19:06:22 GMT
expires: -1
cache-control: private, max-age=0
content-type: text/html; charset=ISO-8859-1
...
```

As we can see, the `curl` request returns multiple headers and the entire HTML body of the response. Since the request went through successfully, the first part of the response is the `200` status code, along with the version of HTTP (this will be HTTP/1.1, HTTP/2, or HTTP/3).

Since this particular request is returning a website, the `content-type` (MIME type) being returned is `text/html`. In a RESTful API, you will likely see `application/json` to indicate the response is JSON.

Interestingly, we can see another type of response by inputting a slightly different URL. Do a `curl` on Google without the `www`.

```
curl -i https://google.com
```

```
HTTP/2 301
location: https://www.google.com/
content-type: text/html; charset=UTF-8
```

Google redirects `google.com` to `www.google.com` and uses a `301` response to indicate that the resource should be redirected.

REST API Endpoints

When an API is created on a server, the data it contains is accessible via endpoints. An **endpoint** is the URL of the request that can accept and process the `GET`, `POST`, `PUT`, or `DELETE` request.

An API URL will consist of the root, path, and optional query string.

- **Root** e.g. `https://api.example.com` or `https://api.example.com/v2`: The root of the API, which may consist of the protocol, domain, and version.
- **Path** e.g. `/users/` or `/users/5`: Unique location of the specific resource.
- **Query Parameters (optional)** e.g. `?location=chicago&age=29`: Optional key value pairs used for sorting, filtering, and pagination.We can put them all together to implement something such as the example below, which would return a list of all users and use a query parameter of `limit` to filter the responses to only include ten results.

```
https://api.example.com/users?limit=10
```

```
https://api.example.com/users?limit=10
```

Generally, when people refer to an API as a RESTful API, they are referring to the naming conventions that go into building API URL endpoints. A few important conventions for a standard RESTful API are as follows:

- **Paths should be plural**: For example, to get the user with an id of `5`, we would use `/users/5`, not `/user/5`.
- **Endpoints should not display the file extension**: Although an API will most likely be returning JSON, the URL should not end in `.json`.
- **Endpoints should use nouns, not verbs**: Words like `add` and `delete` should not appear in a REST URL. In order to add a new user, you would simply send a `POST` request to `/users`, not something like `/users/add`. The API should be developed to handle multiple types of requests to the same URL.
- **Paths are case sensitive and should be written in lowercase with hyphens as opposed to underscores**.

All of these conventions are guidelines, as there are no strict REST standards to follow. However, using these guidelines will make your API consistent, familiar, and easy to read and understand.

REST Alternatives

REST is a great tool, but there are some alternatives that could help in certain cases.

SOAP

SOAP (Simple Object Access Protocol) is an API created in 1998 that was very popular before REST. There are a few major differences. First, SOAP is much more restrictive with the format of responses. Second, SOAP uses XML rather than JSON, which can be helpful for legacy applications but is often larger and more complex than the equivalent JSON. Finally, while SOAP works well with HTTP, it also supports protocols like SMTP.

GraphQL

GraphQL is a newer API format created by Facebook that aims to reduce the number of HTTP requests needed to get data by allowing the client to tell the server exactly what data it needs. Instead of using URL paths, GraphQL has a custom syntax for defining what data the client needs, so that the client gets exactly what they want in one request.

For a great intro to GraphQL, check out this course on GraphQL.

Conclusion

In this article, we learned what REST and RESTful APIs are, how HTTP request methods and response codes work, the structure of an API URL, and common RESTful API conventions. In the next tutorial, we will learn how to put all this theory to use by setting up an Express server with Node.js and building our own API.

*This post has been updated with contributions from **Jacob Jackson**. Jacob is a web developer, technical writer, freelancer, and open-source contributor.*

Tania Rascia
Web Developer/Chicago

I'm Tania, a designer, developer, writer, and former chef. I love creating open source projects and contributing to the developer community. I write the missing instruction manuals of the web.

taniarascia

FEED LIKE FOLLOW

Advertisement

LOOKING FOR SOMETHING TO HELP KICK START YOUR NEXT PROJECT?

QUICK LINKS - Explore popular categories

ENVATO TUTS+

About Envato Tuts+
Terms of Use
Advertise

JOIN OUR COMMUNITY

Teach at Envato Tuts+
Translate for Envato Tuts+
Forums

HELP

FAQ
Help Center