

Guide to Axios Headers

by Ziad Shamndy Dec 26, 2024 #HTTP #NodeJS    



When working with APIs, request and response headers can play a crucial role in ensuring smooth communication between the client and the server.

[Axios](#) is the most popular HTTP client package in javascript and provides a simple and intuitive way to manage headers in your API requests.

In this article, we'll overview axios headers in requests and responses, how to configure them effectively, and explore some practical examples, tips and tricks.

Understanding Axios Headers

Configuring Headers in Axios

Adding Headers to a Single Request

Setting Default Headers

Dynamic Headers

Getting and Inspecting Headers

Setting Headers After Axios Instance Creation

Common Use Cases for Axios Headers

Authentication

Content Negotiation

Custom Headers

What Are the Alternatives to Axios?

Why Choose Axios Over Alternatives?

Power-Up with Scrapfly

FAQ

Conclusion

JOIN THE NEWSLETTER

Get monthly web scraping insights 🖱️

[Learn at ScrapFly Academy](#)

Understanding Axios Headers

Headers are an essential part of HTTP requests and responses, carrying metadata such as content types, authorization tokens, and custom data.

In axios requests, headers often play role in authenticating or for `POST` requests headers are used to provide extra information about posted body like content-type. For axios responses, headers often contain important extra information like what type of data was returned or your connection status like remaining connection quota.

Here are some common headers you might encounter:

Header	Description	Example
<code>Content-Type</code>	Specifies the media type of the request body	<code>application/json</code>
<code>Authorization</code>	Includes credentials for authentication	<code>Bearer <token></code>
<code>Accept</code>	Indicates the response format	<code>application/json</code>
<code>X-</code> prefix	Custom headers	<code>X-Something: value</code>

Why Headers Are Important

Headers go beyond being mere metadata, they are pivotal in defining the behavior and security of API requests. Here's why they matter:

- **Enable Secure Communication:** Headers like `Authorization` facilitate authentication, ensuring sensitive operations are protected.
- **Support Advanced Features:** They enable API functionalities like pagination, versioning, and rate-limiting.
- **Bridge Client and Server Understanding:** Headers clarify expectations, such as content formats (`Content-Type` and `Accept`), to prevent errors and ensure compatibility.

Configuring Headers in Axios

Efficiently managing headers in Axios can significantly enhance your API interactions. Whether you need to configure headers for a single request, set them globally, or update them dynamically, Axios makes it straightforward.

Adding Headers to a Single Request

You can specify headers for an individual request by including them in the `headers` object.

Here's an example for axios get with headers:

```
import axios from "axios";

axios
  .get("https://httpbin.dev", {
    headers: {
      Authorization: "Bearer YOUR_TOKEN",
      "Content-Type": "application/json",
    },
  })
  .then((response) => console.log(response.data))
  .catch((error) => console.error(error));
```

Here's an example for axios post with headers:

```
import axios from "axios";

axios
  .post("https://httpbin.dev", {
    data: "value",
    headers: {
      Authorization: "Bearer YOUR_TOKEN",
    },
  })
  .then((response) => console.log(response.data))
  .catch((error) => console.error(error));
```

So using the `headers` parameter we can specify any headers in all request types.

Setting Default Headers

For headers that should apply to all requests, you can define defaults using Axios to set headers global configuration:

```
axios.defaults.headers.common["Authorization"] = "Bearer YOUR_TOKEN";
axios.defaults.headers.get["User-Agent"] = "me";
axios.defaults.headers.post["Content-Type"] = "application/json";
```

This sets axios get headers with default `User-Agent` value, axios post headers with default `Content-Type` value, and a common header for all requests.

Note that you should be careful with default headers if you're sending requests to more than one origin as there's risk of leaking sensitive information.

Dynamic Headers

Sometimes, headers need to change based on runtime conditions, such as updated tokens. You can update headers dynamically before sending requests.

```
const token = getAuthToken(); // Assume this fetches a token dynamically
axios.defaults.headers.common["Authorization"] = `Bearer ${token}`;
```

This approach dynamically updates the `Authorization` header with a token fetched at runtime. It's especially useful for scenarios like user authentication, where tokens may change frequently.

Getting and Inspecting Headers

You can inspect headers from a server response using the `.headers` property.

```
axios
  .get("https://httpbin.dev")
  .then((response) => {
    console.log("Response Headers:", response.headers);
  })
  .catch((error) => console.error(error));
```

The `response.headers` property lets you access headers returned by the server. This is helpful for debugging or extracting metadata, such as rate limits or API versioning information.

Setting Headers After Axios Instance Creation

If you’re using an Axios instance, you can modify headers on the fly.

```
const apiClient = axios.create({
  baseURL: "https://httpbin.dev",
});

apiClient.defaults.headers.common["Authorization"] = "Bearer YOUR_TOKEN";
```

Creating an Axios instance allows you to configure headers specifically for requests made through that instance. By using interceptors, you can dynamically update headers before each request, ensuring they are always up-to-date.

Common Use Cases for Axios Headers

Headers are essential for tailoring your API requests to meet specific requirements. Below are some common scenarios where headers play a pivotal role, along with code examples and explanations.

Authentication

Headers are critical for secure API access, as they often carry tokens or credentials necessary for authenticating requests.

```
axios.post(
  "https://web-scraping.dev/login",
  {
    username: "user",
    password: "pass",
  },
  {
    headers: {
      Authorization: "Basic <encoded-credentials>",
    },
  }
);
```

the `Authorization` header sends encoded credentials for basic authentication. This is commonly used in login requests or when accessing protected API endpoints.

Content Negotiation

Define the format of the response you expect from the server by specifying the `Accept` header.

```
axios.get("https://httpbin.dev", {
  headers: {
    Accept: "application/json",
  },
});
```

The `Accept` header tells the server that the client expects a response in JSON format. This ensures compatibility between the server’s response and the client’s expectations.

Custom Headers

Custom headers allow you to send additional metadata or special instructions to the server.

```
axios.post(
  "https://httpbin.dev",
  { data: "value" },
  {
    headers: {
      "X-Custom-Header": "custom-value",
    },
  }
);
```

The `X-Custom-Header` is a user-defined header sent to the server. Custom headers are useful for implementing non-standard functionality or passing application-specific metadata.

By understanding these use cases, you can make your Axios requests more powerful, efficient, and aligned with your API’s requirements.

Tired of scraper blocking?

Try Web Scraping API for Free

What Are the Alternatives to Axios?

While Axios is a popular choice for HTTP requests, there are several alternatives that may suit specific needs better. This section explores the most notable options: **Fetch API**, **SuperAgent**, and **Got**, comparing their features to help you make an informed choice.

Fetch API

[Fetch](#) is a native JavaScript API for making HTTP requests. It is lightweight and requires no installation, but it lacks some of the advanced features that Axios provides.

Feature	Axios	Fetch API
Automatic JSON Parsing	Yes	No (requires <code>response.json()</code>)
Global Configuration	Yes (via <code>axios.defaults</code>)	No (manual setup required)
HTTP Interceptors	Yes	No
Error Handling	Centralized and built-in	Requires manual handling (<code>response.ok</code>)
Browser Support	All modern browsers	All modern browsers
Node.js Support	Built-in	Requires polyfill (e.g., <code>node-fetch</code>)

Why Choose Fetch?

Fetch is ideal for lightweight projects where advanced features like interceptors and global configurations are unnecessary. It’s also natively supported by modern browsers, making it a good choice for simpler setups.

You can learn more about Axios vs Fetch in our dedicated article:

Axios vs Fetch: Which HTTP Client to Choose in JS?

we'll explore Fetch vs Axios — compare their key differences, pros, cons, and help you determine which option is best for your project.



SuperAgent

[SuperAgent](#) is a versatile HTTP client with a chainable syntax that simplifies request building.

Feature	Axios	SuperAgent
Popularity	High	Medium
Header Management	Simple with defaults and interceptors	Easy via method chaining
Extensibility	High	Moderate

Why Choose SuperAgent?

SuperAgent is great for developers who prefer a minimalistic approach with clean, chainable methods. However, it lacks the rich ecosystem and community support of Axios, which may make it less suitable for large or complex applications.

Example with SuperAgent:

```
const superagent = require("superagent");

superagent
  .get("https://api.example.com/data")
  .set("Authorization", "Bearer YOUR_TOKEN")
  .then((response) => console.log(response.body))
  .catch((error) => console.error(error));
```

Got

[Got](#) is a high-performance HTTP client designed for Node.js. It excels in customization and efficiency, making it a strong choice for server-side applications.

Feature	Axios	Got
Target Audience	Browser and Node.js	Primarily Node.js
Promise-based API	Yes	Yes
Customization	High	Very High

Why Choose Got?

Got is perfect for Node.js environments where performance and customization are critical. Its rich API supports features like retries, streams, and hooks for complex workflows.

Summary

Here's a quick summary to help you decide:

Criteria	Axios	Fetch API	SuperAgent	Got
Ease of Use	High	Medium	High (via chaining)	Medium
Advanced Features	Extensive (interceptors, defaults)	Limited	Moderate	Extensive
Browser Compatibility	Excellent	Excellent	Moderate	Not applicable
Node.js Support	Excellent	Requires polyfill	Excellent	Excellent

Each of these alternatives has its strengths, and the best choice depends on your specific project requirements. For browser-based applications, Axios or Fetch may be the best fit. For Node.js, Got is a powerful contender. If you prefer chainable methods, SuperAgent might be worth considering.

Why Choose Axios Over Alternatives?

Axios offers several advantages over Fetch and other HTTP clients, making it a strong choice for many use cases. Below are the key reasons why developers prefer Axios:

- Pre-configured Defaults:** Allows you to set global defaults for headers, base URLs, and other configurations.
Reduces redundancy by applying these settings to all requests automatically.
- Automatic Transformations:** Converts request data (e.g., objects to JSON) and parses response data (e.g., JSON to JavaScript objects) out of the box.
Saves time by eliminating manual data transformations.
- Interceptors for Middleware:** Add logic before requests are sent (e.g., adding dynamic headers like tokens).
Modify responses before they are returned to your application.
Simplifies logging, debugging, and error handling.
- Enhanced Error Handling:** Provides centralized and consistent error messages for HTTP request failures.
Makes error debugging easier by capturing status codes, headers, and more.
- Broad Platform Support:** Works seamlessly in both **browser** and **Node.js** environments.
Enables cross-platform development without needing extra polyfills or libraries.
- Convenience Methods:**
Offers shorthand methods (`axios.get` , `axios.post` , etc.) for making common HTTP requests.
Improves readability and reduces boilerplate code compared to Fetch.

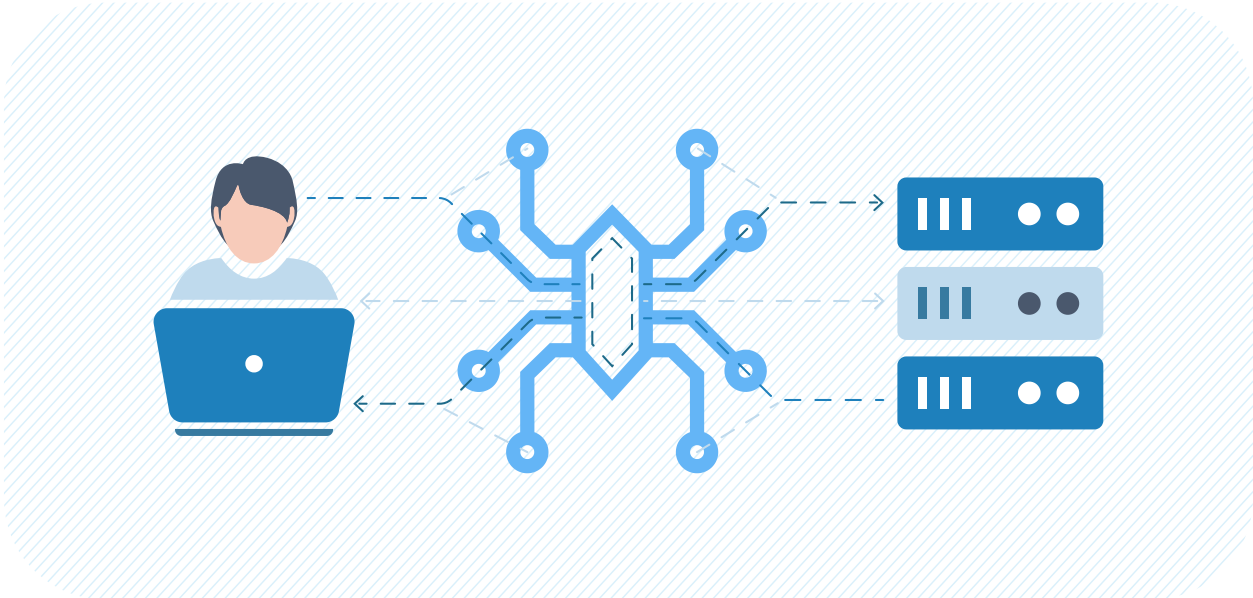
By addressing common pain points in HTTP requests, Axios makes API communication more efficient and developer-friendly.

Power-Up with Scrapfly

ScrapFly provides [web scraping](#), [screenshot](#), and [extraction](#) APIs for data collection at scale.

- [Anti-bot protection bypass](#) - scrape web pages without blocking!

- [Rotating residential proxies](#) - prevent IP address and geographic blocks.
- [JavaScript rendering](#) - scrape dynamic web pages through cloud browsers.
- [Full browser automation](#) - control browsers to scroll, input and click on objects.
- [Format conversion](#) - scrape as HTML, JSON, Text, or Markdown.
- [Python](#) and [Typescript](#) SDKs, as well as [Scrapy](#) and [no-code tool integrations](#).



Try for FREE!

[More on Scrapfly](#)

FAQ

To wrap up this guide, here are answers to some frequently asked questions about Axios Headers.

Does Axios support setting multiple headers with the same name?

No, Axios does not support multiple headers with the same name. The last value set for a header will overwrite any previous values.

How do I handle CORS-related headers in Axios?

To handle CORS (Cross-Origin Resource Sharing), ensure the server is configured correctly. If you need to include credentials (e.g., cookies), set the `withCredentials` property:

```
axios.get("/api/data", { withCredentials: true });
```

Does Axios include default headers in requests?

Yes, Axios includes default headers for common content types like JSON (`application/json`) and form data (`application/x-www-form-urlencoded`). You can also set custom default headers using `axios.defaults.headers` .

Conclusion

Axios simplifies HTTP requests with features like powerful header management, interceptors, and error handling. While alternatives like Fetch have their place, Axios remains the go-to choice for developers seeking robust and feature-rich HTTP client capabilities. Whether you're building APIs, integrating authentication, or handling complex data transformations, mastering Axios headers will elevate your development workflow.

Discover ScrapFly

Try ScrapFly for FREE!

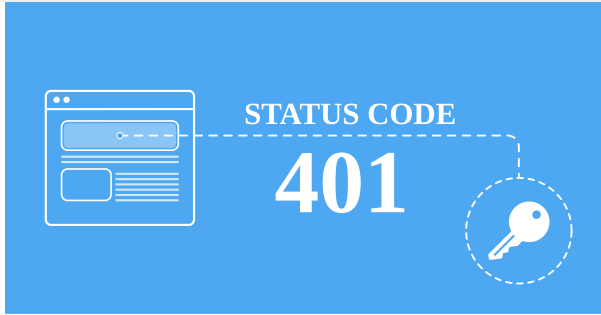
Related Questions

- How to Copy as cURL With Safari?
- How to Copy as cURL With Firefox?
- How To Copy as cURL With Google Chrome?
- What case should HTTP headers be in? Lowercase or Pascal-Case?
- Python httpx vs requests vs aiohttp - key differences
- How to use proxies with PHP Guzzle?
- How to use proxies with NodeJS axios?
- What is Asynchronous Web Scraping?
- How to Copy as cURL With Brave?
- How to Copy as cURL With Edge?
- What Python libraries support HTTP2?
- What are some PhantomJS alternatives for automating browsers?
- How to install mitmproxy certificate on Chrome and Chromium?
- How to use proxies with Python httpx?
- How to add headers to every or some scrapy requests?

More >

HTTP NODEJS    

Related Posts



Dec 20, 2024

What is HTTP 401 Error and How to Fix it

Discover the HTTP 401 error meaning, its causes, and solutions in this comprehensive guide. Learn how 401 unauthorized errors occur.

HTTP



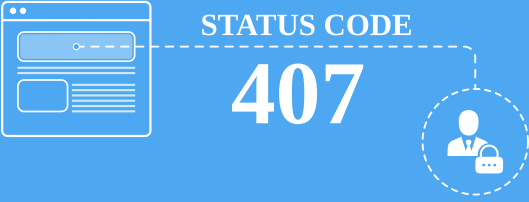
Dec 05, 2024

Comprehensive Guide to OkHttp for Java and Kotlin

Learn how to simplify network communication in Java and Android applications using OkHttp.

HTTP

TOOLS



Dec 03, 2024

What is HTTP 407 Status Code and How to Fix it

Learn everything about the HTTP 407 Proxy Authentication Required error. Understand its causes, including

Company

- Careers
- Terms of service
- Privacy Policy
- Data Processing Agreement
- KYC Compliance
- Status

Integrations

- Zapier
- Make
- N8n
- LlamaIndex
- LangChain

Social



Tools

- Convert cURL commands to Python code
- JA3/TLS Fingerprint
- HTTP2 Fingerprint
- Xpath/CSS Selector Tester

Resources

- API Documentation
- Web Scraping Academy
- Is Web Scraping Legal?
- Web Scraping Tools
- FAQ

Learn Web Scraping

- Web Scraping with Python
- Web Scraping with PHP
- Web Scraping with Ruby
- Web Scraping with R
- Web Scraping with NodeJS
- Web Scraping with Python Scrapy
- How to Scrape without getting blocked tutorial
- Web Scraping with Python and BeautifulSoup
- Web Scraping with Nodejs and Puppeteer
- How To Scrape GraphQL
- Best Proxies for Web Scraping
- Top 5 Best Residential Proxies

Usage

- What is Web Scraping used for?
- Web Scraping for AI Training
- Web Scraping for Compliance
- Web Scraping for eCommerce
- Web Scraping for Finance
- Web Scraping for Fraud Detection
- Web Scraping for Jobs
- Web Scraping for Lead Generation
- Web Scraping for News & Media
- Web Scraping for Real Estate
- Web Scraping for SERP & SEO
- Web Scraping for Social Media
- Web Scraping for Travel

© 2025 Scrapfly - The Best Web Scraping API For Developers