Holberton Coding School - Puerto Rico

RESTful API

Willie J Touset Castellar

C25 - Ponce

Github: https://github.com/neonbloo25

I - What is RESTful API?

Overview

In this guide, you'll learn how to use curl, a command-line tool that allows you to interact with web servers and APIs by sending and receiving data. Whether you're retrieving information from a website, making API requests, or sending data to a server, curl is an essential tool for developers, system administrators, and anyone working with web services. This guide focuses on using curl to perform basic actions such as fetching data, sending requests, and working with APIs directly from the command line.

Objective

By the end of this exercise, you will be able to:

- Install and use curl on your system.
- Perform basic HTTP requests to retrieve and send data using curl, including making GET and POST requests.
- Understand and interpret the responses from APIs, including working with JSON data and response headers.
- Use curl effectively to interact with RESTful APIs, test endpoints, and troubleshoot server responses.

II - Task 0 - Basics of HTTP/HTTPS

Differentiating HTTP and HTTPS

1. Definition

HTTP: Hypertext Transfer Protocol is a protocol used for transmitting data between a client (e.g., web browser) and a server. It is the foundation of most data exchange on the web but does not include encryption.

HTTPS: Hypertext Transfer Protocol Secure is an extension of HTTP, which uses encryption through SSL/TLS protocols to secure data exchanges between the client and server.

2. Security

HTTP: Data is transmitted in plain text, meaning it's vulnerable to eavesdropping, tampering, and man-in-the-middle attacks.

HTTPS: Encrypts data during transmission, making it much harder for hackers to intercept or alter the information being exchanged.

3. Port Number

HTTP: Uses port 80 by default.

HTTPS: Uses port 443 by default.

4. SSL/TLS Encryption

HTTP: Does not use SSL/TLS encryption, meaning the communication is unprotected.

HTTPS: Relies on SSL (Secure Sockets Layer) or TLS (Transport Layer Security) to encrypt the communication, ensuring privacy and integrity.

5. Use Cases

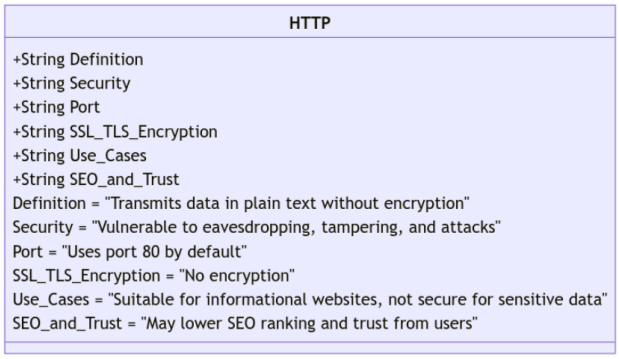
HTTP: Suitable for websites where security is not critical, such as blogs or informational pages. However, it's not recommended for handling sensitive data.

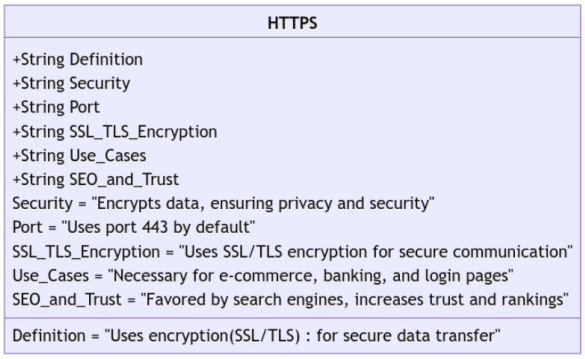
HTTPS: Essential for any website that handles sensitive information, such as e-commerce sites, online banking, or login pages, where user data needs to be protected.

6. SEO and Trust

HTTP: Modern search engines, like Google, may rank HTTP websites lower due to security concerns. Additionally, users may not trust sites that lack HTTPS.

HTTPS: Websites using HTTPS are generally favored by search engines and are seen as more trustworthy by users, which can lead to better rankings and user confidence.





Understanding HTTP Structure

HTTP (Hypertext Transfer Protocol) is how web browsers (clients) communicate with servers to request and receive web content (like HTML pages, images, or data).

1. HTTP Request

When a client (like a web browser) asks for something from the server, it sends an HTTP request. Here are the key parts:

- Request Line: It tells the server what to do:

- **GET:** Get something from the server.
- **POST:** Send something to the server (like form data).
- **PUT:** Update something on the server.
- **DELETE:** Remove something on the server.
- **URL:** The location of the resource (e.g., `/home`).
- **HTTP Version:** Specifies the HTTP version (e.g., `HTTP/1.1`).

- Headers: Extra details about the request (like what kind of response the client can handle, the browser version, etc.).

- Body (optional): Data the client is sending (e.g., when filling out a form).

2. HTTP Response

After processing the request, the server sends a response back to the client. Key parts:
- Status Line: Tells if the request was successful and provides a code (like `200 OK` or `404 Not Found`).

- Headers: Additional info (e.g., type of content, length of data, etc.).

- Body: The actual content the server is sending back (e.g., HTML, images, JSON).

3. Common HTTP Status Codes

- 200 OK: Everything is fine, and the requested data is sent.
- 404 Not Found: The requested resource isn't available.
- 500 Internal Server Error: Something went wrong on the server.

4. HTTP Methods

Different actions you can ask the server to do:

- GET: Get data from the server.

- POST: Send data to the server.

- PUT: Update data on the server.

- DELETE: Delete data on the server.

5. Cookies

Servers can store small pieces of data (cookies) on the client's side. This can help with:
- Remembering who you are (e.g., logging in).
- Storing preferences.

6. Security

- HTTPS: A secure version of HTTP using encryption to protect data.
- HTTP Headers for Security: There are special headers to protect against attacks like hacking.

7. Caching

Web browsers or servers can store copies of data to improve speed. Cache-related headers tell the client how to store data (e.g., how long to keep it before checking for updates).

Exploring HTTP Methods and Status Codes

Step 1: HTTP Methods

HTTP methods tell the server what kind of action the browser wants to perform. Here are 4 common HTTP methods that you can list:

1. GET

Description: Requests data from the server.

Use case: Fetching a webpage, retrieving data from an API, loading images or other resources.

Example: When you open a webpage like https://www.example.com, your browser sends a GET request to retrieve the page content.

2. POST

Description: Sends data to the server (usually to submit information like a form).

Use case: Submitting a login form or a comment on a blog.

Example: When you log in to a website, you usually send your username and password via a POST request.

3. PUT

Description: Updates existing data on the server.

Use case: Changing your profile details or updating a record in a database.

Example: Updating your email address on a social media platform might send a PUT request to the server.

4. DELETE

Description: Deletes data from the server.

Use case: Removing a post or deleting your account on a website.

Example: If you delete a comment from a forum, a DELETE request might be sent to the server.

Step 2: HTTP Status Codes

HTTP status codes are the server's way of telling your browser what happened with the request. Here are 5 common status codes you can list:

1. 200 OK

Description: The request was successful, and the server is returning the requested data.
Use case: This is the typical response you get when you successfully load a webpage.

2. 404 Not Found

Description: The requested page or resource cannot be found on the server.
Use case: Happens when you try to visit a webpage that doesn't exist, like typing in an incorrect URL.

3. 500 Internal Server Error

Description: Something went wrong on the server, and it can't complete the request.
Use case: Occurs when a website’s server has an issue, and it can’t handle your request properly.

4. 301 Moved Permanently

Description: The resource you're looking for has been permanently moved to a new location.
Use case: Happens when a website changes its URL, and you get automatically redirected to the new one.

5. 403 Forbidden

Description: The server understands your request, but it's refusing to fulfill it (you don’t have permission).
Use case: Happens when you try to access a restricted area of a website without the proper credentials.

III - Task 1 Consume data from an API using command line tools (curl)

Overview

Use `curl` to interact with APIs and fetch or send data directly from the command line. `curl` is a powerful tool that allows you to make HTTP requests to web servers, APIs, and websites, making it essential for testing and debugging.

1. Install curl

Before you can start using `curl`, you need to install it on your system.

Check Installation:

To confirm that `curl` is installed, run the following command:

bash

>>> curl --version

If installed, you'll see details about the version of `curl` and supported protocols.

2. Fetch Content from a Webpage

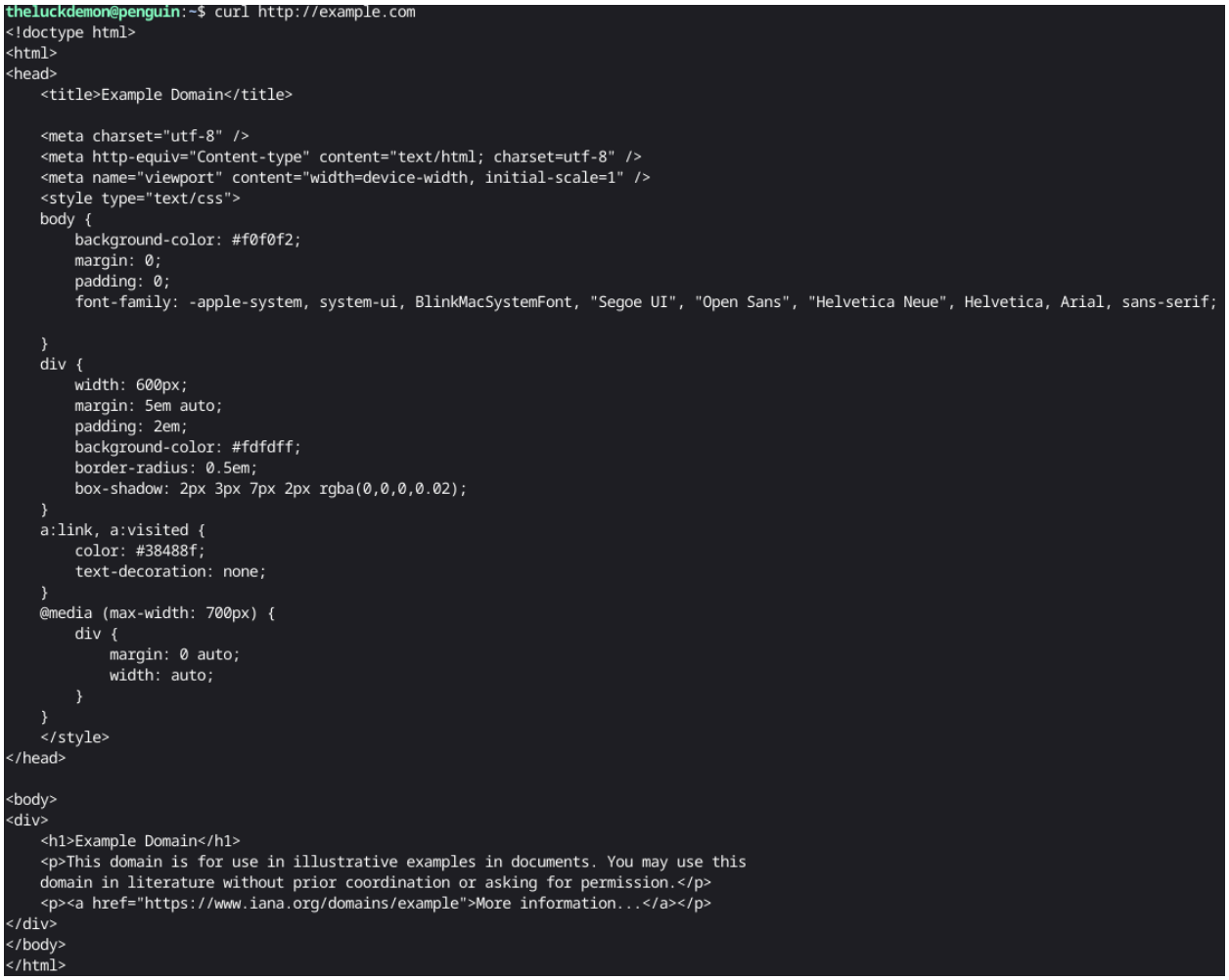
Use `curl` to fetch data from a website or an API endpoint.

For example, to fetch the content of a webpage, you can run:

bash

>>> curl http://example.com

This command will return the HTML content of the page.



```
theluckdemon@penguin:~$ curl http://example.com
<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
    @media (max-width: 700px) {
        div {
            margin: 0 auto;
            width: auto;
        }
    }
    </style>
</head>

<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is for use in illustrative examples in documents. You may use this
    domain in literature without prior coordination or asking for permission.</p>
    <p><a href="https://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

3. Fetch Data from an API

To fetch data from a public API. Use JSONPlaceholder, a free API for testing and prototyping.

To fetch a list of posts, run the following command:

bash

>>> curl https://jsonplaceholder.typicode.com/posts

This will return data in JSON format, which typically includes an array of posts, each containing details like `id`, `title`, `body`, and `userId`.


```
  {
    "userId": 5,
    "id": 46,
    "title": "aut quo modi neque nostrum ducimus",
    "body": "voluptatem quisquam iste\nvoluptatibus natus officiis facilis dolorem\nquis quas ipsam\nvel et voluptatu
m in aliquid"
  },
```


4. Fetch Only Headers

In some situations, you may only want to see the headers (metadata) of a response, such as the status code or server information, without retrieving the entire body content.

To fetch only the headers for the same API request, run:

bash

>>> curl -I https://jsonplaceholder.typicode.com/posts

The `-I` flag tells `curl` to only retrieve the response headers, which could include important info like:

- HTTP status code (e.g., `200 OK`)
- Content type
- Server details


```
]theluckdemon@penguin:~$curl -I https://jsonplaceholder.typicode.com/posts
HTTP/2 200
date: Fri, 21 Feb 2025 18:15:33 GMT
content-type: application/json; charset=utf-8
report-to: {"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1735023079&sid=e11707d5-02a7-43ef-b45e-2cf4d2036f7d&s=hL%2FBKWzMNyFLdsKUxcAaPEP9dvol22oIojOSJMi9AIA%3D"}]}
reporting-endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1735023079&sid=e11707d5-02a7-43ef-b45e-2cf4d2036f7d&s=hL%2FBKWzMNyFLdsKUxcAaPEP9dvol22oIojOSJMi9AIA%3D
nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"response_headers":["Via"]}
x-powered-by: Express
x-ratelimit-limit: 1000
x-ratelimit-remaining: 999
x-ratelimit-reset: 1735023107
vary: Origin, Accept-Encoding
access-control-allow-credentials: true
cache-control: max-age=43200
pragma: no-cache
expires: -1
x-content-type-options: nosniff
etag: W/"6b80-Ybsq/K6GwwqrYkAsFxqDXGC7DoM"
via: 1.1 vegur
cf-cache-status: HIT
age: 22450
server: cloudflare
cf-ray: 9158b8f2aea044e1-ATL
alt-svc: h3=":443"; ma=86400
server-timing: cfL4;desc="?proto=TCP&rtt=110743&min_rtt=84421&rtt_var=84302&sent=7&recv=8&lost=0&retrans=1&sent_bytes=3465&recv_bytes=764&delivery_rate=14130&cwnd=252&unsent_bytes=0&cid=f3161ba85000a0ca&ts=329&x=0"
```

5. Make a POST Request

To send data to an API, you can make a **POST request**. This is commonly used to create or update data on the server.

For example, to create a new post on the **JSONPlaceholder** API, use the following command:

bash

>>> curl -X POST -d "title=foo&body=bar&userId=1" https://jsonplaceholder.typicode.com/posts

Explanation of flags:

- `-X POST`: Specifies that the request method is POST.

- `-d`: Sends data with the request (the data is in the form of key-value pairs, in this case, `title`, `body`, and `userId`).

The server will simulate creating a new post and will return a response, typically with a `created post` object, including an ID.

```
theluckdemon@penguin:~$ curl -X POST -d "title=foo&body=bar&userId=1" https://jsonplaceholder.typicode.com/posts
{
  "title": "foo",
  "body": "bar",
  "userId": "1",
  "id": 101
```

Helpful Tips:

- Headers-only request (`-I`): Use this flag when you only want to check the status of a request (e.g., to troubleshoot or check server settings) without downloading the entire content.

- Request Type (`-X`): The `-X` flag lets you specify the HTTP method (GET, POST, PUT, etc.). By default, `curl` makes GET requests, but you can change this behavior using `-X` for other methods like POST or PUT.

- Data with POST (`-d`): When you want to send data (e.g., form submissions or JSON data), use the `-d` flag with `curl`. This is useful for sending information to create or update resources on an API.

Expected Output:

1. Version Check: Running `curl --version` will show you the installed version of `curl` and the protocols it supports.

2. Fetching Posts: Running `curl https://jsonplaceholder.typicode.com/posts` will display an array of JSON-formatted posts.

3. Headers-Only Request: Running `curl -I https://jsonplaceholder.typicode.com/posts` will show response headers without content.

4. POST Request: Running `curl -X POST -d "title=foo&body=bar&userId=1" https://jsonplaceholder.typicode.com/posts` will simulate creating a new post, and the response will contain the post with an ID.

With these basic `curl` commands, you can start interacting with APIs directly from the command line, fetching data, inspecting headers, and sending data. Whether you are testing an API or automating tasks, `curl` is an invaluable tool for developers and system administrators.

IV-Conclusion

Mastering curl equips you with a powerful tool for testing and interacting with web services directly from the command line. By learning to fetch data, send requests, and interpret responses, you'll be able to quickly diagnose issues, prototype applications, and automate interactions with APIs. Whether you're working on web development, server management, or automation, curl is a versatile and essential skill in any developer’s toolkit.