Assignment:

Secure Coding Practice:

Image Generated by Dalle-E 2 from chat GPT

Prepared for

Kokub Sultan

Prepared by Group 6

Valentine Jingwa, Jean-Pierre Nde-Forgwang

Security System, Cohort E

School of Advance Digital Technology

SAIT

31 January 2024

Index

Introduction to HTTP:

Origins of HTTP

The HyperText Transfer Protocol (HTTP) was developed by Tim Berners-Lee at CERN in 1989-1990. HTTP was designed to facilitate communication between web browsers and servers, enabling the transfer of hypertext documents on the World Wide Web (WWW). It became the foundation of data communication for the web, with the first documented version, HTTP/0.9, being a simple protocol for raw data transfer across the Internet [1].

How HTTP Works

HTTP operates as a request-response protocol in the client-server computing model. A web browser or client sends an HTTP request to the server, where the requested resources (such as HTML files, images, etc.) are hosted. The server then processes the request and responds back to the client with the requested information or an error message if the resource cannot be found or accessed. HTTP messages are composed of headers and a body. The headers contain metadata about the request or response, while the body contains the actual data being transmitted [1].

Alternatives to HTTP

While HTTP is the most widely used protocol for web traffic, there are alternatives designed for specific use cases, including:

HTTPS (HTTP Secure)

An extension of HTTP that uses SSL/TLS to encrypt the communication, enhancing security.

FTP (File Transfer Protocol)

Used for transferring files between clients and servers, suitable for large files.

WebSocket

A protocol providing full-duplex communication channels over a single TCP connection, designed for real-time data transfer between a browser and a server.

SPDY (deprecated)

Developed by Google to transport web content more efficiently, it laid the groundwork for HTTP/2.

The Stateless Nature of HTTP

HTTP is inherently stateless, meaning each request-response pair is independent; the server does not retain session information between different requests from the same client. This simplifies the server design but necessitates mechanisms like cookies, session storage, or tokens for maintaining stateful sessions for applications like shopping carts or user logins [3].

The Difference Between HTTP and HTTPS

The primary difference between HTTP and HTTPS is the use of encryption. HTTPS employs SSL/TLS to encrypt the communication between client and server, ensuring data integrity, confidentiality, and authentication. This prevents

eavesdroppers from understanding the intercepted communications, making HTTPS essential for protecting sensitive transactions [2].

Importance of HTTP with Examples

HTTP is crucial for the functioning of the web. It enables the structured publication and retrieval of webpages, forming the backbone of information exchange on the Internet. For example:

- **Web Browsing:** Every time you visit a website, your browser uses HTTP or HTTPS to request the web pages from a server.
- **APIs and Web Services:** Many web services provide APIs over HTTP, allowing for data exchange between different software systems. For instance, RESTful APIs use HTTP methods explicitly for creating, reading, updating, and deleting resources.
- **Social Media:** Platforms like Facebook, Twitter, and Instagram rely on HTTP/HTTPS to deliver content and services to users, enabling actions like posting, commenting, and streaming.

HTTP's simplicity and extensibility have allowed it to evolve and support the vast and complex web ecosystem we have today, from simple websites to sophisticated web applications delivering rich multimedia content and interactive experiences.

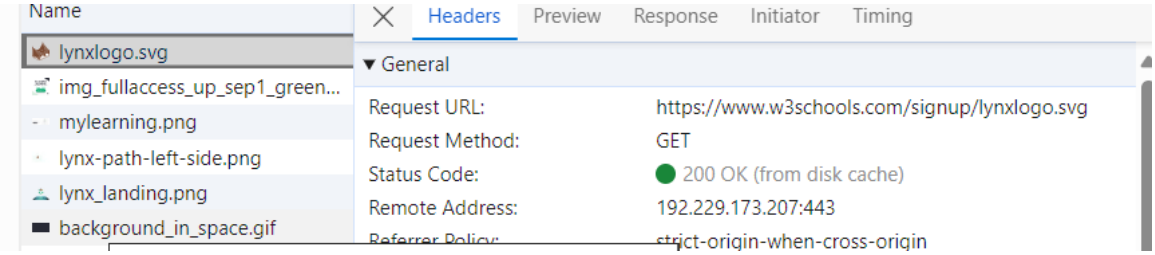
Exploration of HTTP Message Types

GET

The GET method requests data from a specified resource without affecting the resource itself. It's idempotent, meaning multiple identical requests should have the same effect as a single one.[4]

Example Use Case: Accessing a webpage. When you type a URL into your browser and press enter, your browser sends a GET request to the server to retrieve the HTML of the webpage.

How to Capture: Open your browser's developer tools (F12 or right-click → "Inspect"), go to the "Network" tab, and navigate to a website. Look for the initial document request, which should be a GET request.



Screenshot Get the request of w3sc

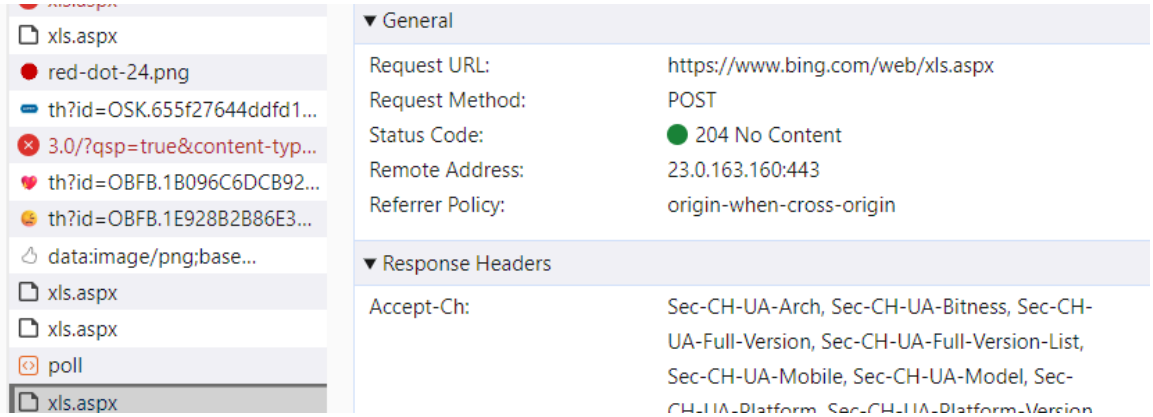
POST

The POST method submits data to be processed to a specified resource, often causing a change in state or side effects on the server.[4]

Example Use Case: Submitting a form on a website, such as a login form. The form data is sent to the server to be processed.

How to Capture: On a website with a form, open the developer tools and navigate to the "Network" tab. Fill out and submit the form to capture the POST

request.



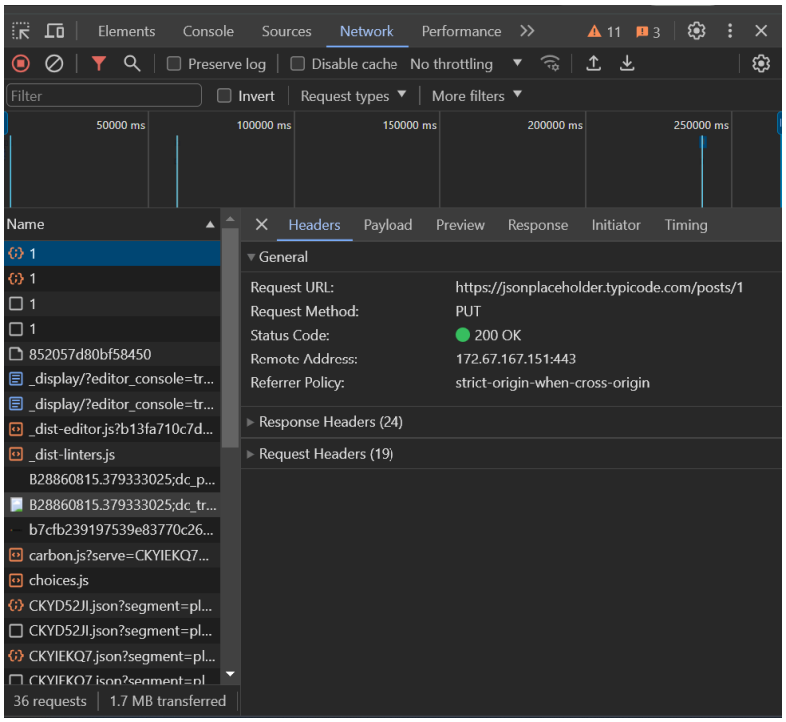
Screenshot Get the request of w3sc

PUT

The PUT method replaces all current representations of the target resource with the request payload.[4]

Example Use Case: Updating a user profile. Unlike POST, PUT is idempotent, meaning that repeating the request will not have further effects after the first.

How to Capture: Use a REST client or a website that allows data manipulation, and observe the PUT request in the developer tools' "Network" tab when updating a resource.

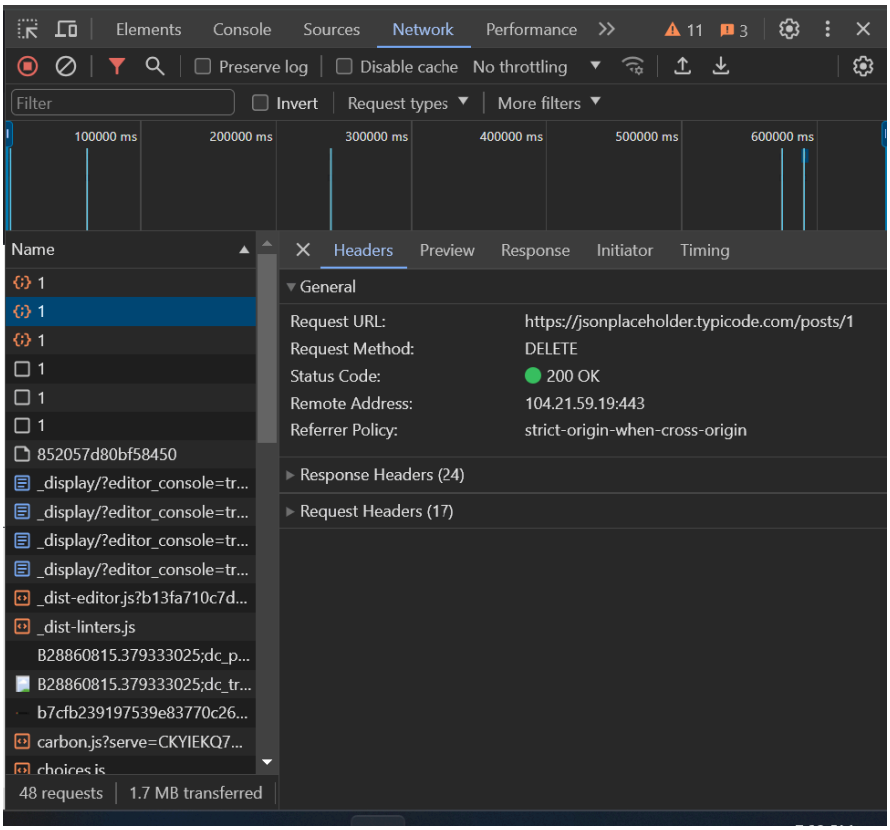


DELETE

The DELETE method removes the specified resource [4].

Example Use Case: Deleting a user account. Sending a DELETE request to the server instructs it to remove the specified resource.

How to Capture: This might be harder to capture in regular web browsing. Instead, using API testing tools like Postman to send a DELETE request to a supported service or API endpoint can demonstrate this.

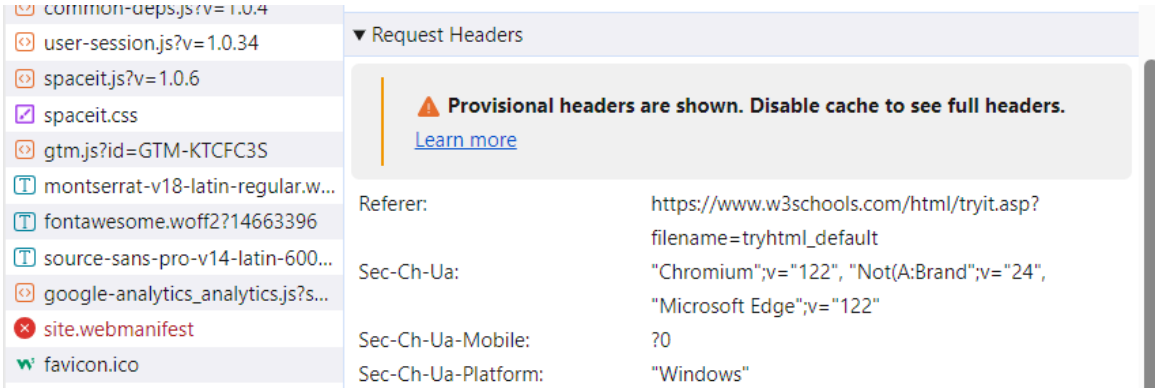


HEAD

Similar to GET, but it returns only the response headers without the response body. Useful for checking what a GET request will return before making the request or checking if a resource exists [4].

Example Use Case: Fetching metadata about a document, such as its size or last modified date, without downloading the document itself.

How to Capture: Some web development tools and REST clients allow you to make HEAD requests, and you can observe the response in the tool's network log.

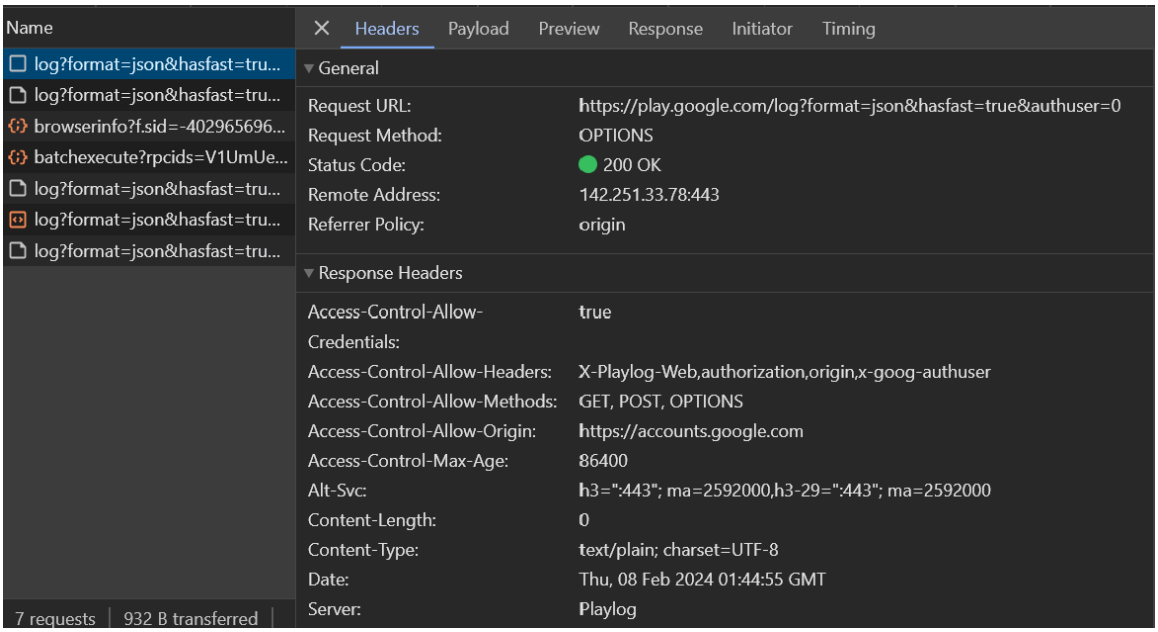


OPTIONS

The OPTIONS method describes the communication options for the target resource, useful for identifying supported HTTP methods or for CORS preflight requests [4].

Example Use Case: Before sending an actual request (like a POST request in a cross-origin scenario), the browser can send an OPTIONS request to check if the CORS policy allows it.

How to Capture: In the developer tools' "Network" tab, observe the OPTIONS request that automatically precedes the actual request in cross-origin scenarios.

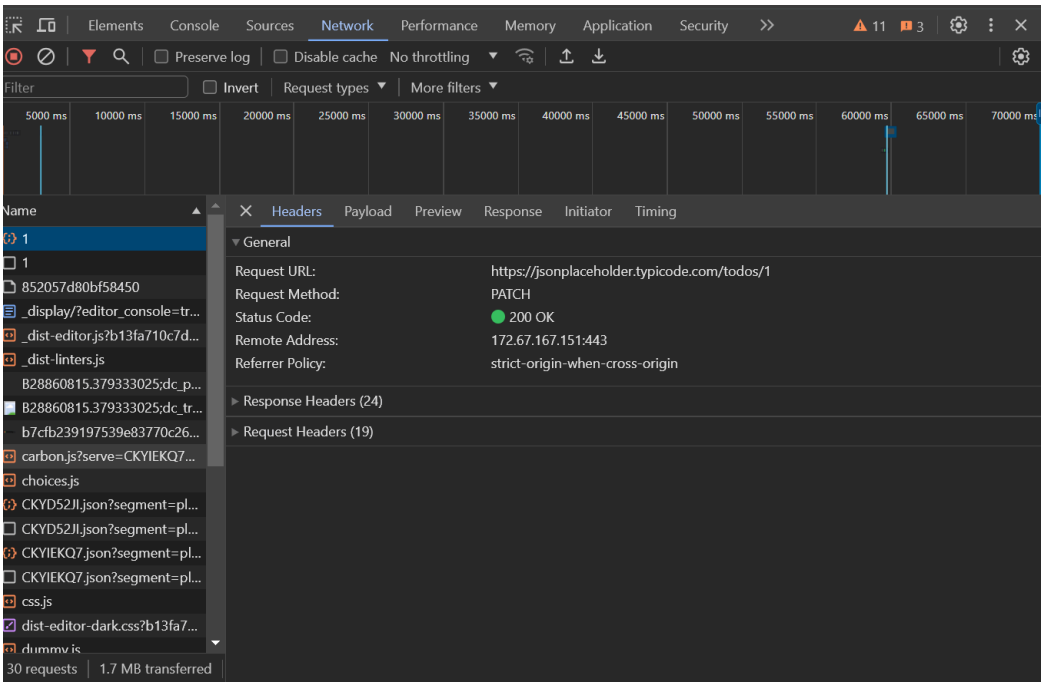


PATCH

The PATCH method applies partial modifications to a resource. It's intended for situations where you only need to update a part of the resource, rather than replacing the whole resource as PUT does [4].

Example Use Case: Updating a user's email address without altering other user data. PATCH requests send only the data that needs to be updated, making it more efficient for small changes.

How to Capture: Similar to PUT and DELETE, capturing a PATCH request might require a REST client or accessing a specific service that supports PATCH operations. You can observe this request through the developer tools or REST client when you make a partial update to a resource.



HTTP Headers Analysis:

HTTP headers play a critical role in the HTTP request and response cycle, providing essential context and instructions that affect how HTTP requests are made and responses are understood and processed. Here's an in-depth look at the specific headers you mentioned [5]:

User-Agent

Purpose: The User-Agent header is sent by the client (usually a web browser or a web crawler) to identify itself to the server. It contains information about the browser type, version, operating system, and sometimes additional details

about the device or software using the web. This information allows servers to tailor content or functionality based on the client's capabilities or preferences. For example, a website might render differently on a mobile browser than on a desktop browser based on the User-Agent string [6].

Content-Type

Purpose: The Content-Type header is used in both HTTP requests and responses. In requests, it tells the server what type of data is being sent by the client. In response, it tells the client what type of data the server is returning. This header is essential for understanding how to interpret the body of the message. For instance, Content-Type: text/html indicates that the body contains HTML code, while Content-Type: application/json indicates JSON formatted data. Properly defining the content type helps ensure that the data is correctly handled and processed by both the client and server [6].

Authorization

Purpose: The Authorization header is used by the client to authenticate itself to the server. It typically contains credentials in the form of encoded username and password pairs, tokens (such as JSON Web Tokens), or other authentication data. This header is critical for controlling access to protected resources. When a server receives a request with an Authorization header, it can validate the credentials and determine if the request should be allowed or denied based on the user's permissions [6].

Cookie

Purpose: The Cookie header is sent by the client to the server and contains stored cookies previously set by the server using the Set-Cookie header.

Cookies are key-value pairs used for managing session state, tracking user behavior, and storing user preferences. Servers use this information for various purposes, such as maintaining user sessions across multiple requests, personalizing content, or tracking user behavior for analytics [6].

Set-Cookie

Purpose: The Set-Cookie header is sent from the server to the client in order to instruct the client to store a cookie. The server includes this header in its response when it needs to set or update a cookie value on the client's device. Each Set-Cookie header contains one cookie with attributes such as its name, value, expiration time, path, domain, and flags (like Secure or HttpOnly). Cookies set by the server can then be included in subsequent requests to the server via the Cookie header, enabling the server to recognize returning users, sessions, or preferences [6].

Analyzing HTTP Headers

To examine these headers in real-world HTTP communication:

Open Developer Tools

Use the developer tools in your web browser (accessible via F12 or right-click → "Inspect").

Network Tab

Navigate to the "Network" tab to monitor HTTP requests and responses.

Trigger Requests

Browse a website or perform actions (like logging in) to generate network activity.

Inspect Headers

Click on individual network requests or responses in the developer tools to view their headers. You can see outgoing **User-Agent**, **Content-Type**, and **Authorization** headers in requests, and incoming Set-Cookie headers in responses. The Cookie header will appear in subsequent requests after cookies have been set.

Report

Overview of HTTP

HTTP, or Hypertext Transfer Protocol, is the backbone of data communication on the World Wide Web. It defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands. When you enter a website in your browser, an HTTP request is sent to the server hosting the site. The server then responds with an HTTP message, along with the content of the website in HTML format for your browser to display. This protocol facilitates a standardized way for computers to communicate with each other, making it possible for us to browse the internet [1].

HTTP Message Types

HTTP operates on a request-response model, where the client sends a message to the server, which returns a response using various methods or "verbs".

- **GET:** Requests data from a specified resource. Example: Accessing a webpage [4].

- **POST:** Submits data to be processed to a specified resource. Example: Submitting a form [4].
- **PUT:** Replaces all current representations of the target resource with the uploaded content. Example: Updating a file [4].
- **DELETE:** Removes the specified resource. Example: Deleting a user account [4].
- **HEAD:** Similar to GET, but it returns only the headers and no body. Used for fetching metadata [4].
- **OPTIONS**: Describes the communication options for the target resource. Useful for CORS (Cross-Origin Resource Sharing) preflight requests [4].
- **PATCH:** The PATCH method applies partial modifications to a resource [4].

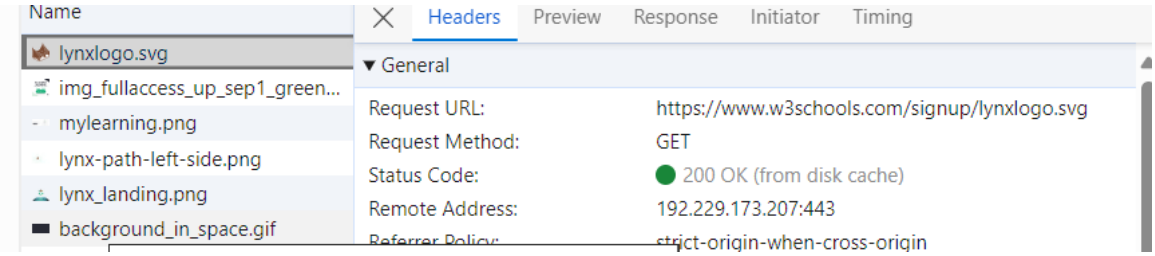
HTTP Headers

HTTP headers are crucial in HTTP requests and responses, providing essential information about the browser or server, requested page, and server response. They include:

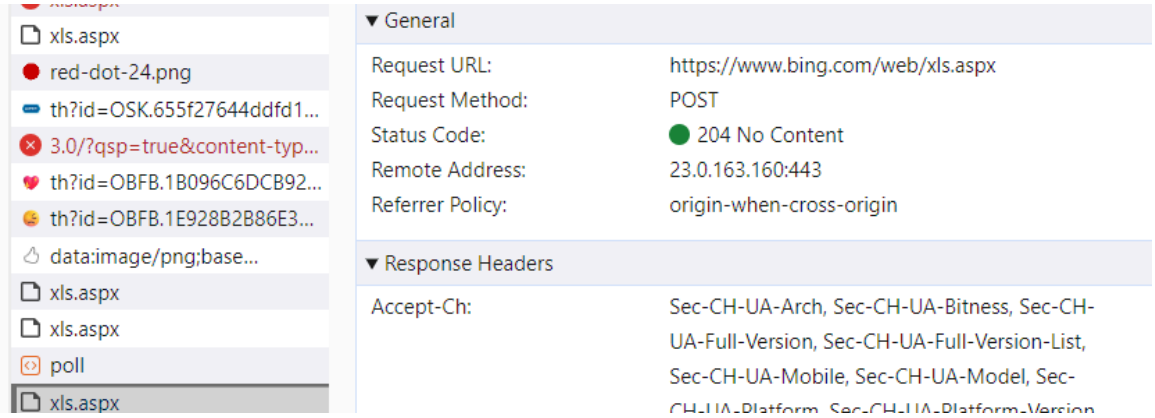
General headers: Apply to both requests and responses but have no relation to the data in the body [7].
Request headers: Contain more information about the resource to be fetched or about the client itself [7].
Response headers: Provide additional information about the response, like its location or server [7].
Entity headers: Contain information about the body of the resource, like content length or MIME type [7].

Practical Observations [8]

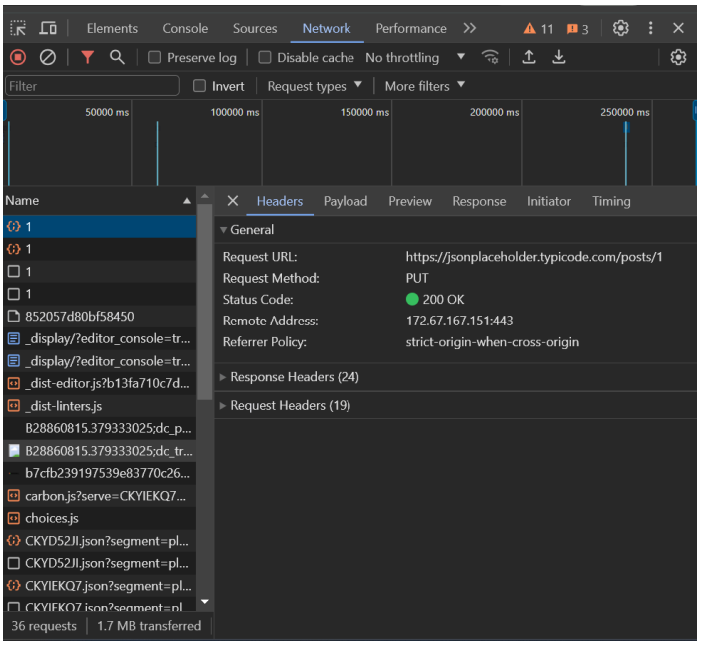
- **GET:**



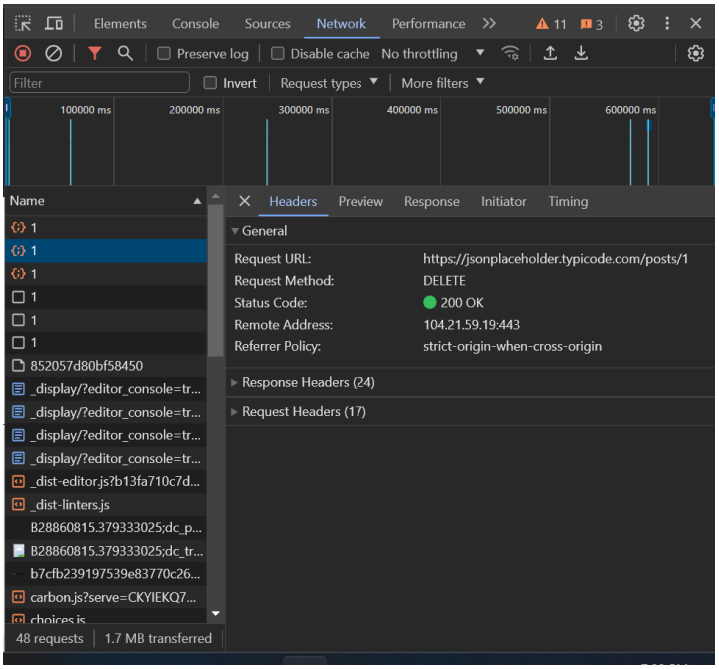
- **POST:**



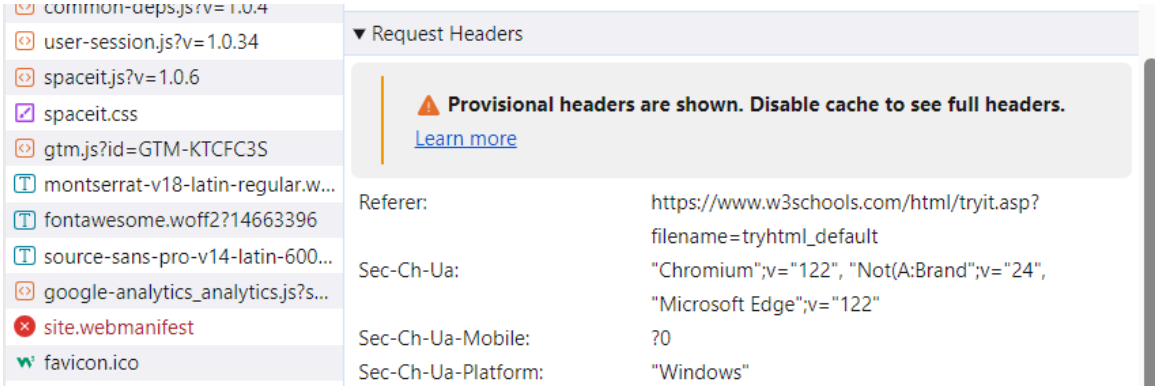
- **PUT:**


Elements
Console
Sources
Network
Performance
Preserve log
Disable cache
No throttling
Filter
Invert
Request types
More filters
50000 ms
100000 ms
150000 ms
200000 ms
250000 ms
Name
Headers
Payload
Preview
Response
Initiator
Timing
General
Request URL: https://jsonplaceholder.typicode.com/posts/1
Request Method: PUT
Status Code: 200 OK
Remote Address: 172.67.167.151:443
Referrer Policy: strict-origin-when-cross-origin
Response Headers (24)
Request Headers (19)
36 requests
1.7 MB transferred

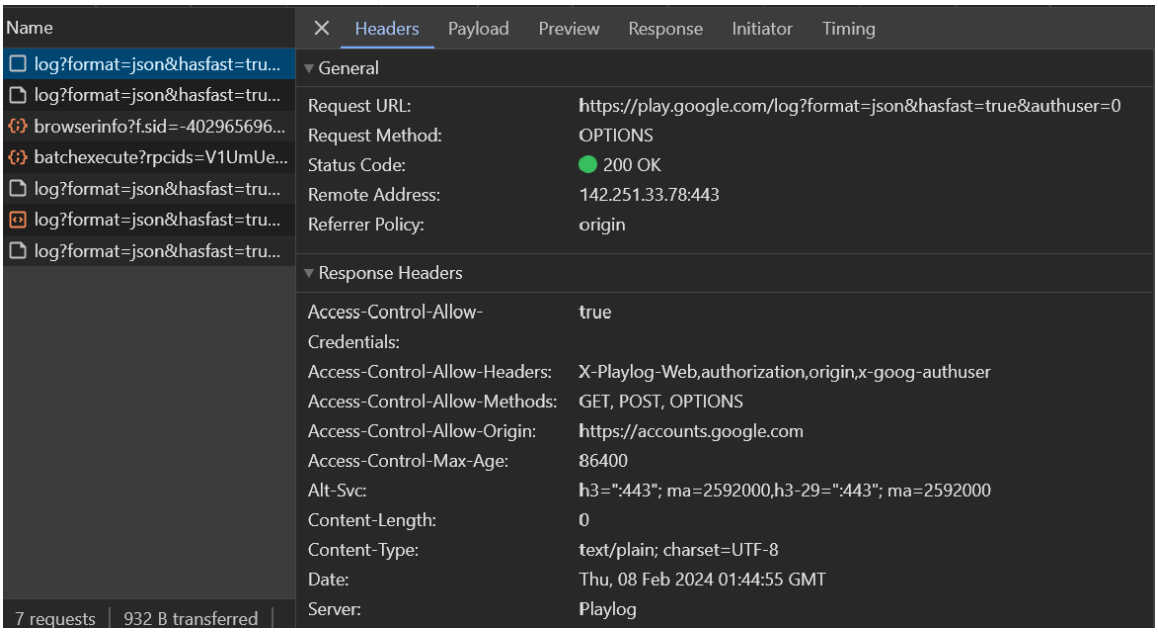
- **DELETE:**


Elements
Console
Sources
Network
Performance
Preserve log
Disable cache
No throttling
Filter
Invert
Request types
More filters
100000 ms
200000 ms
300000 ms
400000 ms
500000 ms
600000 ms
Name
Headers
Preview
Response
Initiator
Timing
General
Request URL: https://jsonplaceholder.typicode.com/posts/1
Request Method: DELETE
Status Code: 200 OK
Remote Address: 104.21.59.19:443
Referrer Policy: strict-origin-when-cross-origin
Response Headers (24)
Request Headers (17)
48 requests
1.7 MB transferred

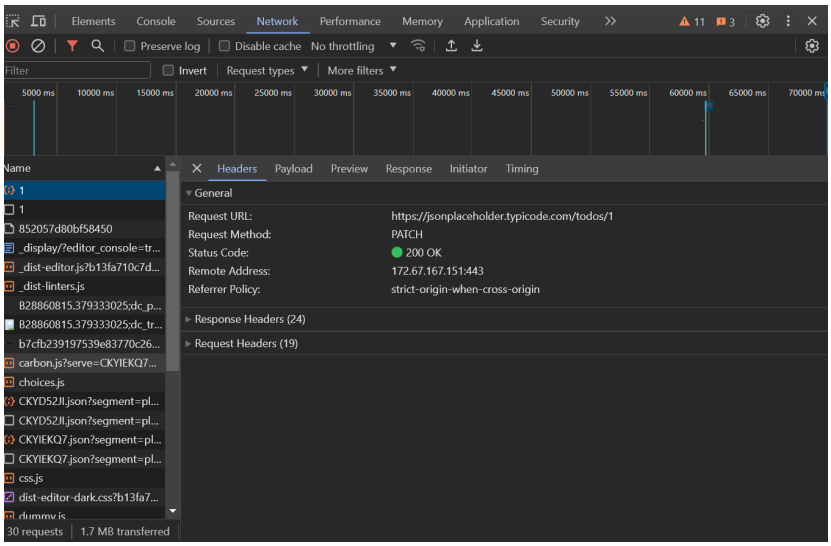
- **HEAD:**



- **OPTIONS**:



- **PATCH:**:



References:

Cite any sources used for research. Ensure that all sources are appropriately referenced using a standard citation format.

1) Wikipedia contributors, "HyperText Transfer," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/HTT. [Accessed: Jan. 17, 2024].
2) Wikipedia contributors, "HyperText Transfer Protocol," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/HTTP. [Accessed: Jan. 17, 2024].
3) Wikipedia contributors, "Stateless protocol," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Stateless_protocol. [Accessed: Jan. 17, 2024].
4) Wikipedia contributors, "HTTP request methods," in HyperText Transfer Protocol, Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/HTTP#Request_methods. [Accessed: Jan. 17, 2024].
5) MDN Web Docs contributors, "HTTP headers," MDN Web Docs. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers. [Accessed: Jan. 17, 2024].
6) Wikipedia contributors, "List of HTTP header fields," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/List_of_HTTP_header_fields. [Accessed: Jan. 17, 2024].
7) GeeksforGeeks contributors, "HTTP Headers," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/http-headers/. [Accessed: Jan. 17, 2024].

8) "JSFiddle," [Online]. Available: https://jsfiddle.net/. [Accessed: Jan. 17, 2024].