HTTP Request Lifestyle

.NET

In the OSI reference model, the communications between a computing system are split into seven different abstraction layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application.

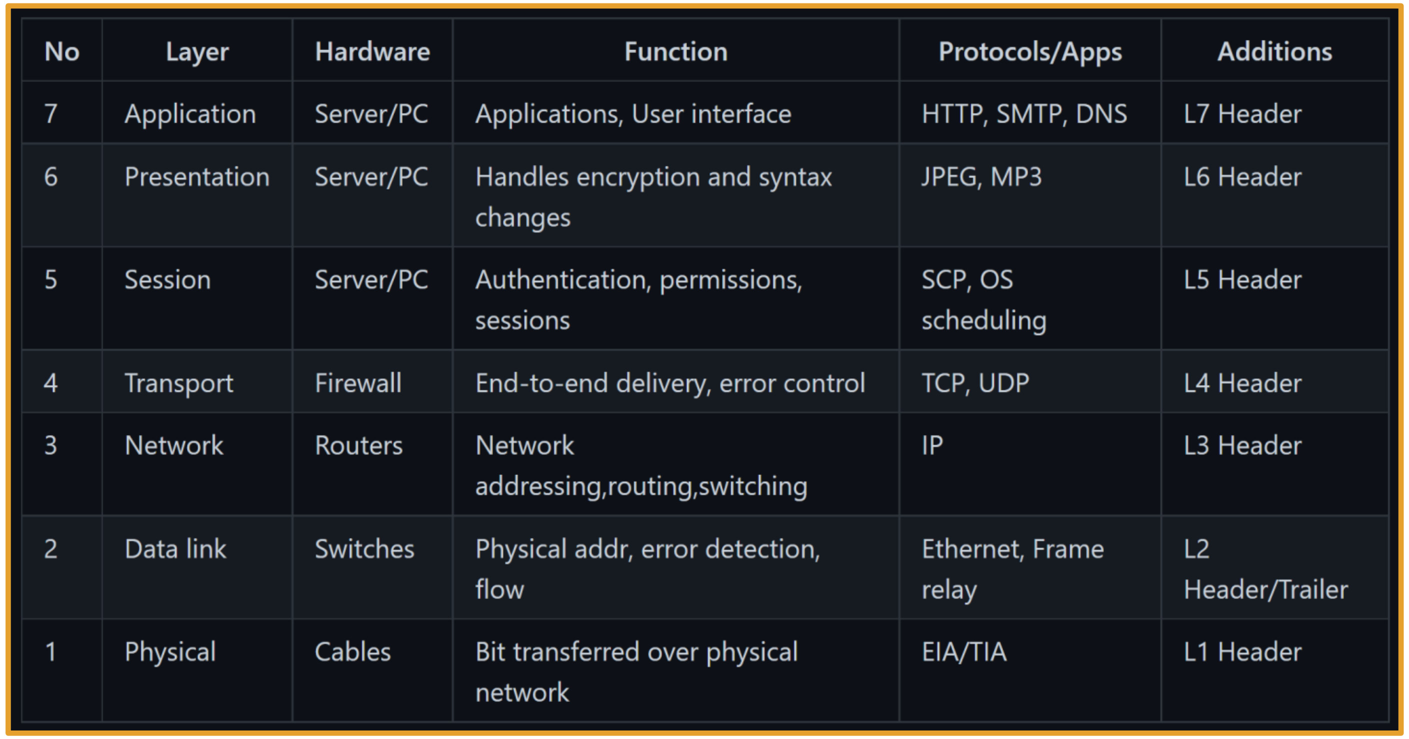
HTTPS://EN.WIKIPEDIA.ORG/WIKI/OSI_MODEL

OSI Model.

https://github.com/hardikvasa/http-connection-lifecycle

The Open Systems Interconnection model (OSI model) is a conceptual model that 'provides a common basis for the coordination of standards for the purpose of systems interconnection'.

In the OSI reference model, the communications between a computing system are split into seven different abstraction layers: ***Physical, Data Link, Network, Transport, Session, Presentation, and Application***.
These OSI layers are present on each device connected to the network. Requests are passed through the layers as they are validated and responded to.



No	Layer	Hardware	Function	Protocols/Apps	Additions
7	Application	Server/PC	Applications, User interface	HTTP, SMTP, DNS	L7 Header
6	Presentation	Server/PC	Handles encryption and syntax changes	JPEG, MP3	L6 Header
5	Session	Server/PC	Authentication, permissions, sessions	SCP, OS scheduling	L5 Header
4	Transport	Firewall	End-to-end delivery, error control	TCP, UDP	L4 Header
3	Network	Routers	Network addressing,routing,switching	IP	L3 Header
2	Data link	Switches	Physical addr, error detection, flow	Ethernet, Frame relay	L2 Header/Trailer
1	Physical	Cables	Bit transferred over physical network	EIA/TIA	L1 Header

HTTP Request Life Cycle Overview

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

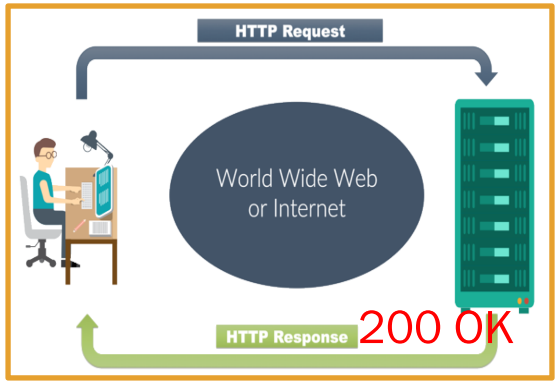
HTTP sessions consist of three basic phases:

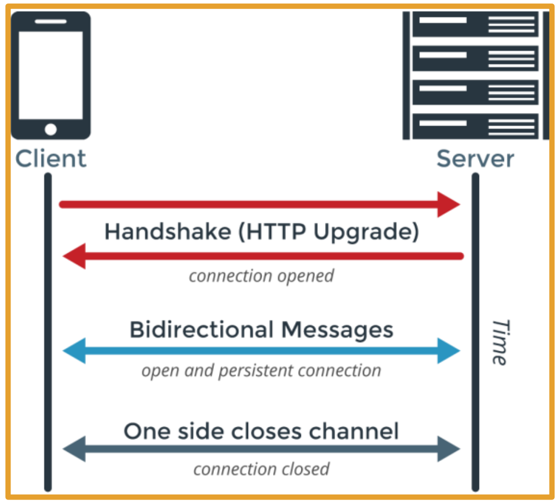
1. The client establishes a TCP (Transmission Control Protocol) connection.
2. The client sends its ***Request*** and waits for the ***Response***.
3. The server processes the request and sends back its ***Response*** with a ***status code*** and appropriate data.

The client-server model requires an explicit ***Request***.

Workarounds to this limitation are:

- ping the server periodically via the XMLHTTPRequest,
- Fetch APIs,
- using the WebSockets API

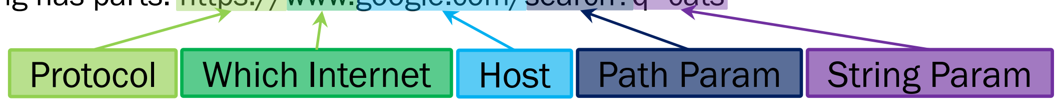




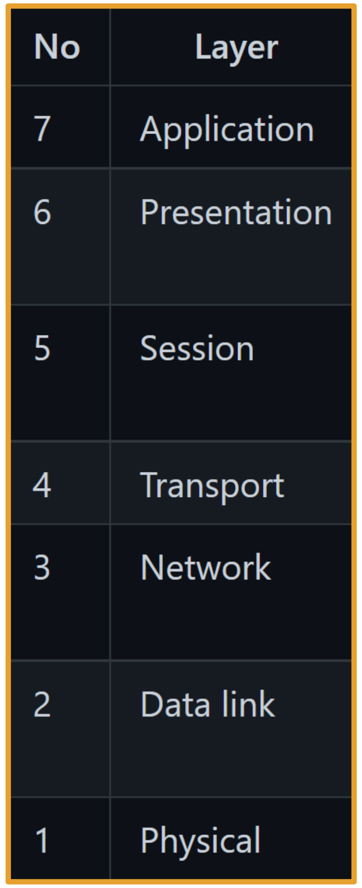
Http Request Steps in depth (1/3)

https://github.com/hardikvasa/http-connection-lifecycle

1. Browser parses the URL. Parse means to divide out the parts that make up the hole.
    1. The query string has parts: https://www.google.com/search?q=cats



2. Address Resolution: The browser searches for an IP address (255.255.255.255) that matches the URL host value (www.revature.com) searched for by the user or app; It searches in multiple locations, in this order.
    1. Local browser cache. **Packets in caches have a Time-to-live (TTL) expiration. After TTL reaches 0, its containing packet is discarded.
    2. Computer Operating System cache
    3. Internet Service Provider (ISP) cache.
    4. Domain Name System (DNS) server.
3. The browser builds the TCP/IP packet by passing it through each ***OSI model*** layer. Each layer adds its own header to the packet. The packet is opened with the headers in opposite order.
4. Steps to route the request through the local router using the Media Access Control (MAC) address. The MAC address identifies the specific device globally.
    1. Get the MAC address using ARP (Address Resolution Protocol) protocol.
    2. The browser connects to the router.

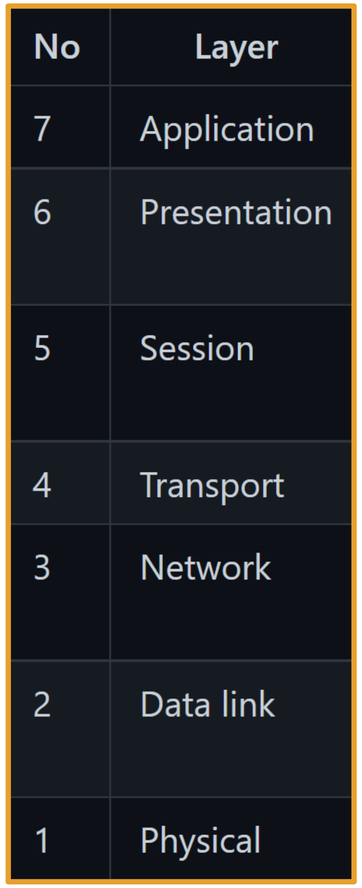


No	Layer
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data link
1	Physical

Http Request Steps in depth (2/3)

https://github.com/hardikvasa/http-connection-lifecycle

5. The uses Network Address Translation (NAT). NAT exists to resolve router uses the designated default route or individual machines when a single IP address serves multiple devices.
6. The request takes multiple jumps from router (node) to router along its path.
    5. Every device has a globally unique MAC address that is included as the destination in layer 2 (Data Link) of the OSI model.
    6. If the MAC destination is the same as the router, the router looks into level 3 (Network) to see if it has a matching MAC address.
        5. If not, the router decrements the query's TTL* and looks into its own routing table for matching routes advertised from other routers.
        6. If unfound, the router does ARP to find the MAC address of the next router in the query's path.
    7. Once found the destination and source addresses are reversed and the query is sent back to the client.
7. Transmission Control Protocol (TCP) 3-way handshake: once the local machine receives a response, it begins content negotiation for Multimedia Message Service (MMS) size, sequence number, ACK (acknowledgment) type, etc.
    5. Each of the client and server will set certain fields in the query, then confirm receipt by incrementing a specific field. This way, each side knows if a message was lost and can request that packet be resent.
    6. Through this process, both client and server have agreed on rules for the data transfer.



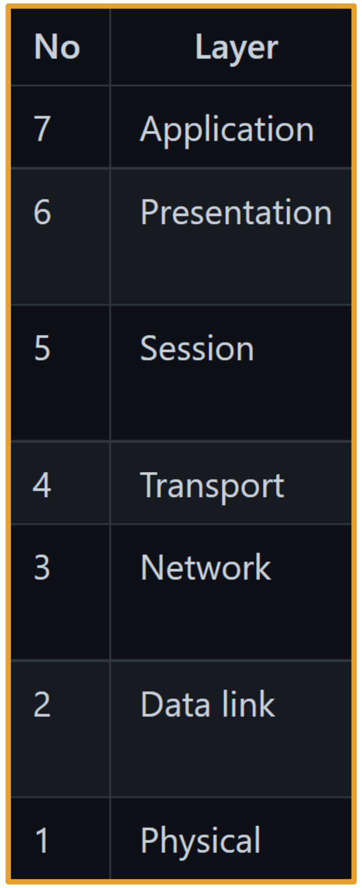
No	Layer
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data link
1	Physical

*When the Time-To-Live value reaches 0, the router discards the packet.

Http Request Steps in depth (3/3)

https://github.com/hardikvasa/http-connection-lifecycle

8. TLS (Transport Layer Security) handshake: This takes place only when connecting to a secure website and is used to agree on the terms of secure communication.
    5. Client sends it’s TLS version, cypher algorithms, and compression type.
    6. Server responds with confirmation that it can match the client’s methods and its Certificate Authority (CA).
    7. Client verifies the CA sent with its list of trusted CA’s
        5. If the CA is verified here, the client sends a string of random bytes and encrypts it along with the server’s public key.
    8. The whole string is used by the server to create a “symmetric (master) key”. The symmetric key is returned.
    9. The client gets the symmetric key and responds with the complete message. It encrypts a hash of the whole transmission up to this point with the symmetric key.
    10. The server creates its own hash of the communication up to this point, decrypts the clients hash to verify that they match.
        5. If they match, the server responds with a final message encrypted with the symmetric key.
9. Process the request: The server receives the request and sends back a response.
    8. The server routes the request to the needed port where the web service is running.
10. Close the connection: When the client sees that the transfer is complete, it sends a connection closing request.

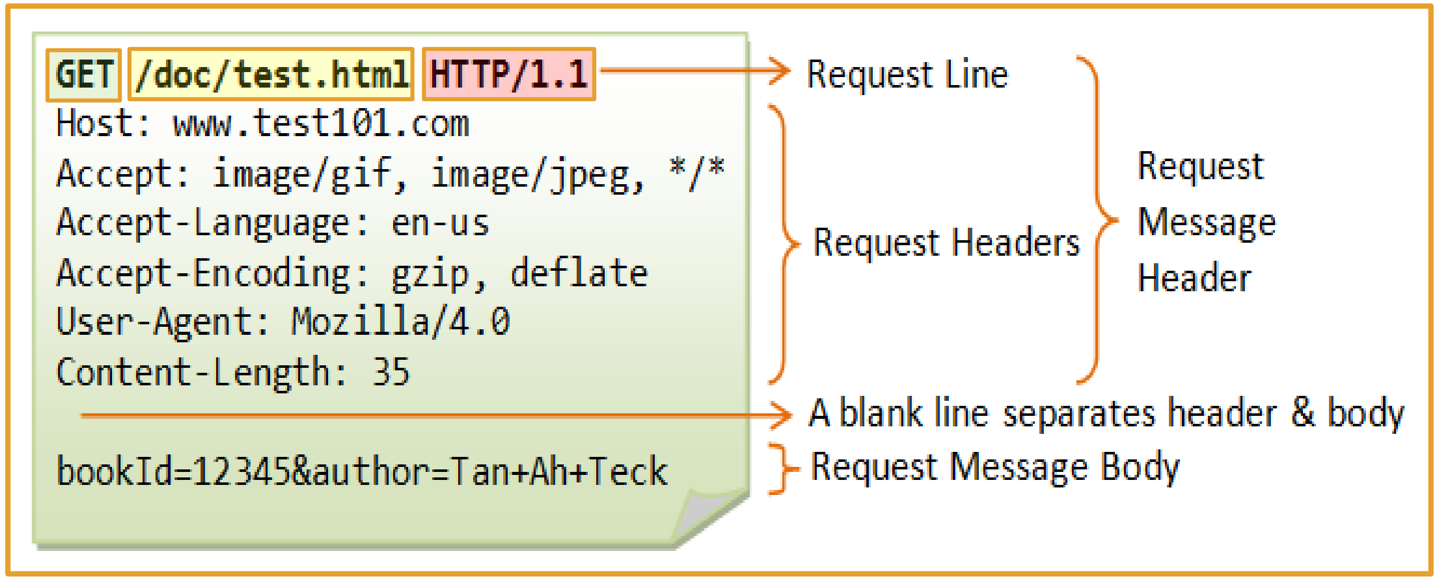


No	Layer
7	Application
6	Presentation
5	Session
4	Transport
3	Network
2	Data link
1	Physical

Connecting and Sending A Request(1/2)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

- The client always establishes the connection.
- The client-server model does not allow the server to send data to the client without an explicit Request.
- The HTTP default port is port:80.
- The URL of a page to fetch contains both the domain name, and the port number.



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

On successful connection, the web browser sends the request. A client request consists of text directives, separated by CRLF (Carriage Return, Line Feed), divided into three blocks: Line 1 - request method followed by the absolute URL doc path without the protocol or domain name and the HTTP protocol version.

Sending A Request (2/2)

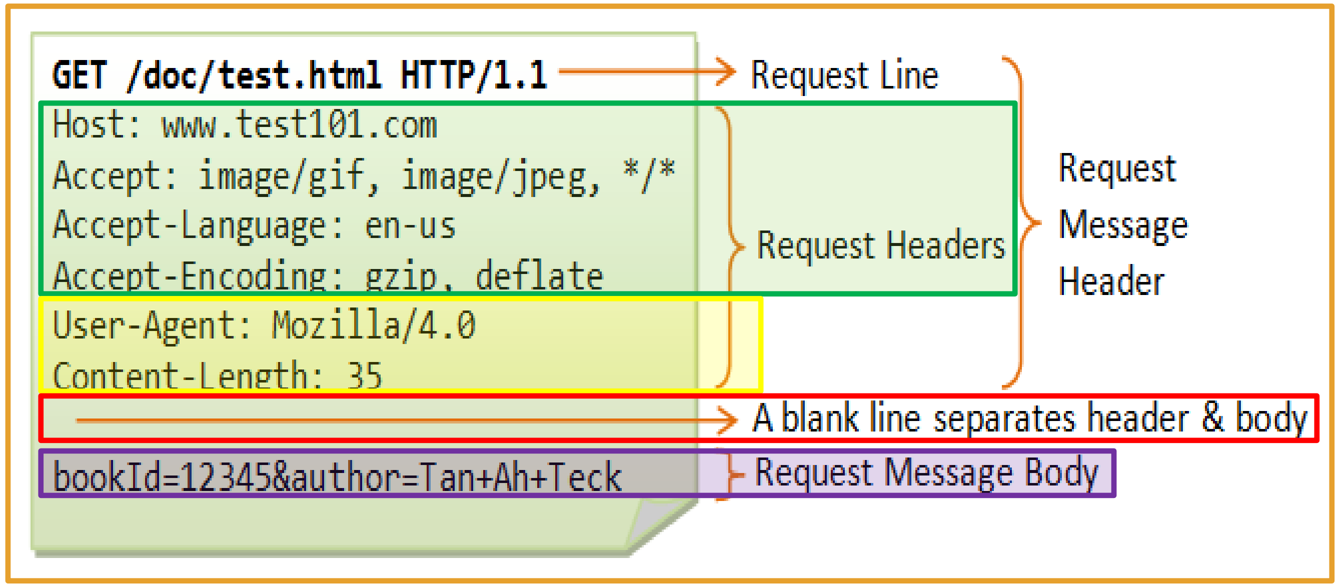
https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

HTTP headers provide the server with:

- information about what type/formats of data is appropriate in the response (e.g., what language, what MIME types, etc),
- other headers which inform and may alter its behavior such as not sending an answer if it is already cached,

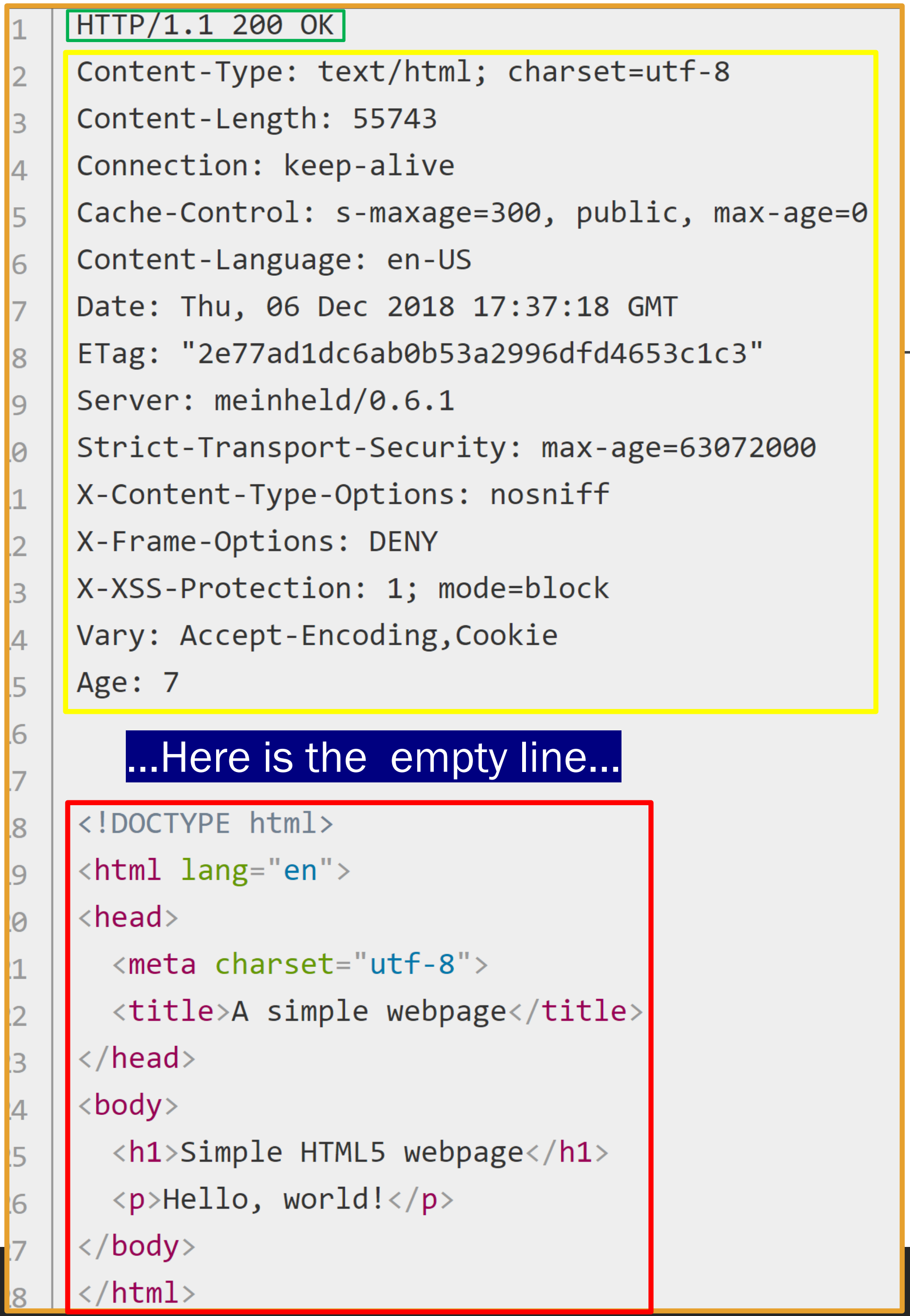
The Header section is followed by:

- An empty line.
- An (optional) **data** block, mainly used by the POST method which contains further data.



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```



```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 55743
Connection: keep-alive
Cache-Control: s-maxage=300, public, max-age=0
Content-Language: en-US
Date: Thu, 06 Dec 2018 17:37:18 GMT
ETag: "2e77ad1dc6ab0b53a2996dfd4653c1c3"
Server: meinheld/0.6.1
Strict-Transport-Security: max-age=63072000
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding,Cookie
Age: 7

...Here is the  empty line...
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A simple webpage</title>
</head>
<body>
  <h1>Simple HTML5 webpage</h1>
  <p>Hello, world!</p>
</body>
</html>
```

Response

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

The server processes the ***Request*** and returns a ***Response***. A server ***Response*** is formed of text directives, separated by CRLF (Carriage Return Line Feed), divided into three blocks:

Line 1 (the status line) :an acknowledgment of the HTTP version used, and a request ***status code*** (and its meaning).

Subsequent lines represent specific HTTP headers, giving the client information like data type and size, compression algorithm used, hints about caching, etc. It ends with an empty line.

The final block is a data block which contains the (optional) data

Additional Resources

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#HTTP_Messages

https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design