HTTP

- Whenever you type a URL into your browser what you are doing is you are actually accessing a web resource using HTTP, the **browser turns the URL into a request message and sends it to the HTTP server.**
- The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message.
- This is the premise of the HTTP request – response cycle.
- A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax: **protocol://hostname:port/path-and-file-name**
- There are 4 parts in a URL:
  1) **Protocol:** The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
  2) **Hostname:** The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
  3) **Port:** The TCP port number that the server is listening for incoming requests from the clients.
  4) **Path-and-file-name:** The name and location of the requested resource, under the server document base directory.

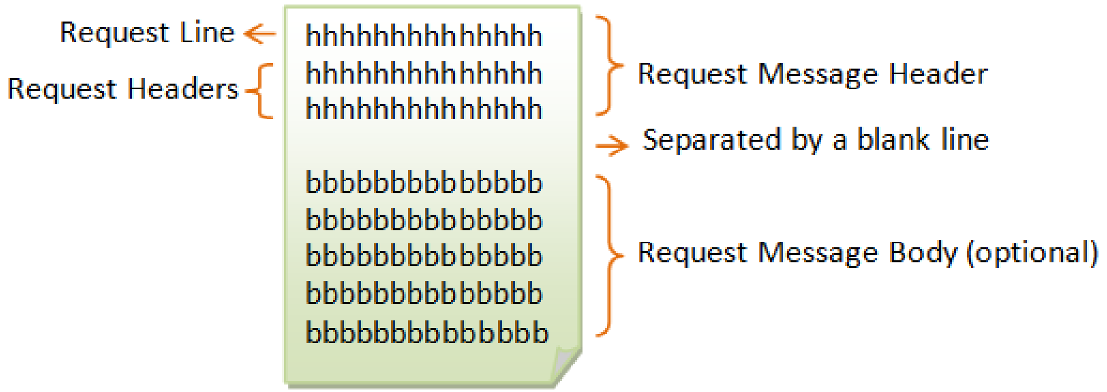
- For example, in the URL http://www.nowhere123.com/docs/index.html,
- the communication **protocol** is HTTP;
- the **hostname** is www.nowhere123.com.
- The **port number** was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP.
- The **path and file name** for the resource to be located is "/docs/index.html".

So what happens when my request reaches the server ?

- Remember the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.
- In its idling state, an HTTP server does nothing but listening to the IP address(es) and port(s) specified in its configuration for incoming requests.
- When this request message reaches the server, the server can take either one of these actions:
  1) The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
  2) The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
  3) The request cannot be satisfied, the server returns an error message.

Structure of HTTP Request and Response Messages

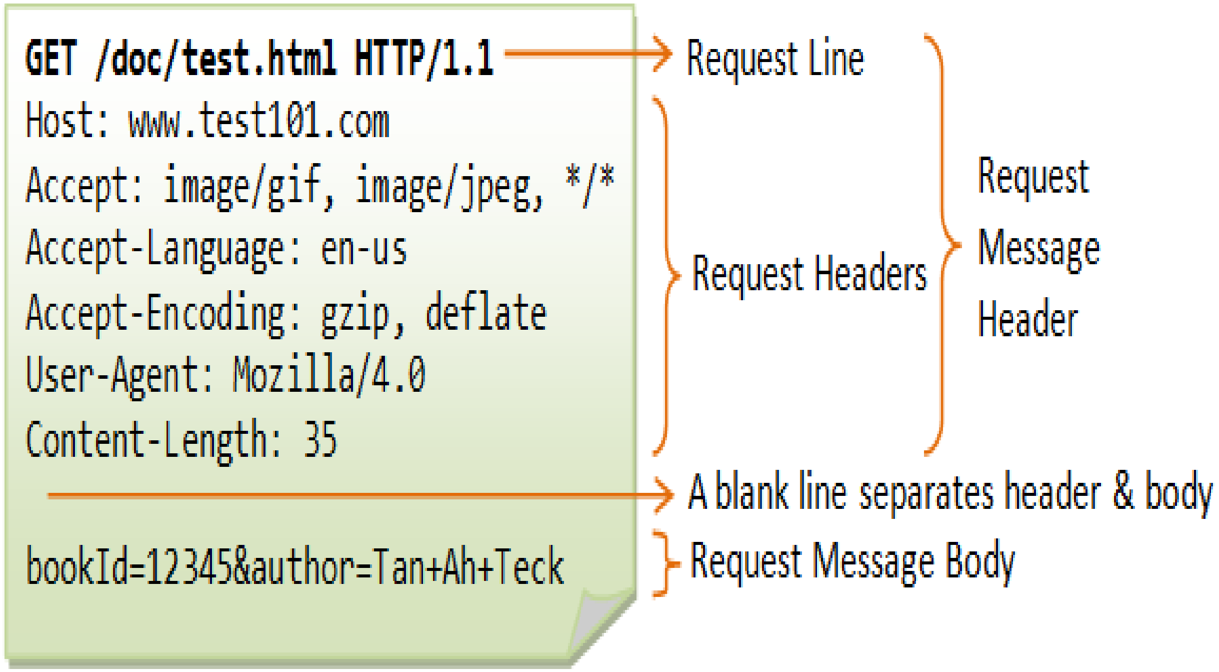
- HTTP is a client-server application-level protocol.
- HTTP client and server communicate by sending text messages.
- The **client sends a request** message to the server.
- The **server, in turn, returns a response** message.
- An **HTTP message** consists of **a message header** and an **optional message body**, separated by a blank line



HTTP Request Message

- The first line of the header is called the request line, followed by optional request headers.
- The request line has the following parts:
- request-method-name: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
- request-URI: specifies the resource requested.
- HTTP-version: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.
- Example of a request line GET /test.html HTTP/1.1

- The request headers are in the form of name:value pairs. Multiple values, separated by commas, can be specified.



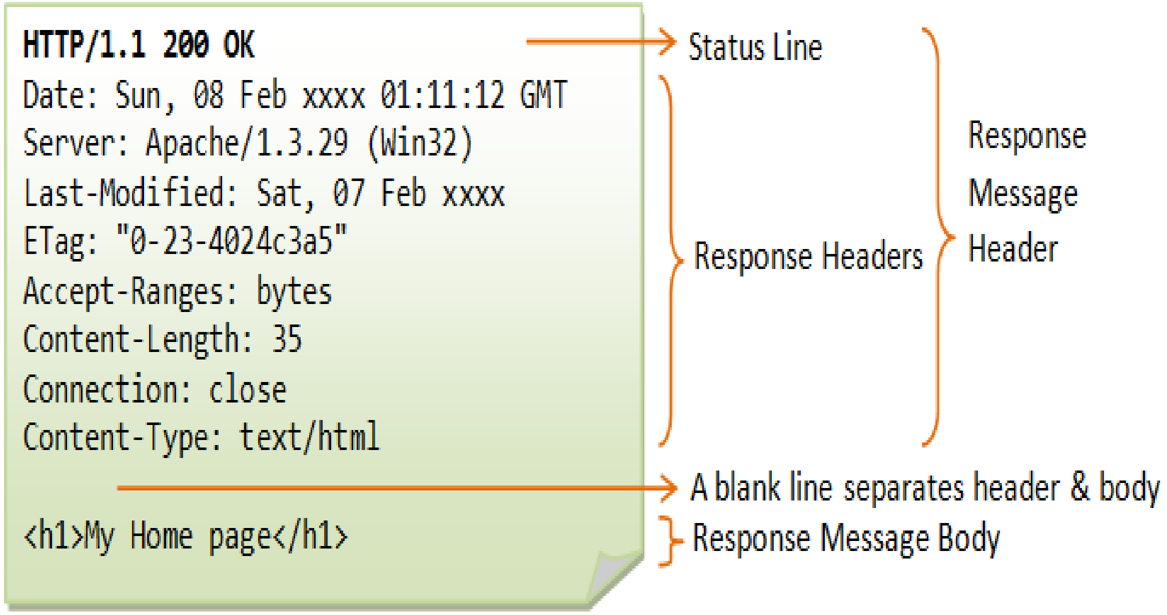
```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

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

HTTP Response Message

- The format of the HTTP response message is as follows:
- status line,
- followed by optional response header(s).
- Separated by a blank line
- Followed by an optional response message body

- The status line has the following syntax:
  HTTP-version status-code reason-phrase
- **HTTP-version:** The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- **status-code**: a 3-digit number generated by the server to reflect the outcome of the request.
- **reason-phrase**: gives a short explanation to the status code.
- Example of a status code and reason phrase include : "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

- The response headers are in the form name:value pairs
- Eg: response-header-name: response-header-value1, response-header-value2,



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

HTTP Request Methods

- HTTP protocol defines a set of request methods.
- CRUD (Create, Read, Update, Delete) operations in web applications are typically mapped to specific HTTP methods as follows:

1. Create (C):

- HTTP Method: **POST**
- Purpose: Used to create a new resource on the server. When you send a POST request, you're typically sending data that the server will use to create a new record or resource.

2 . Read (R):

- HTTP Method: **GET**
- Purpose: Used to retrieve data from the server. When you send a GET request, you're typically asking the server to provide you with the data associated with a specific resource or a list of resources.

3. Update (U):

- HTTP Method: **PUT or PATCH**
- Purpose:
  - PUT: Used to update an entire resource or create it if it doesn't exist. When you send a PUT request, you typically send the entire updated resource, including fields that might not have changed.
  - PATCH: Used to update a resource partially, meaning you only send the fields that have changed, and the server updates those fields accordingly.

4. Delete (D):

- HTTP Method: **DELETE**
- Purpose: Used to request the removal of a resource from the server. When you send a DELETE request, you're typically asking the server to delete the resource identified by the request URL.

- Other HTTP request methods include:
- **HEAD:** A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- **TRACE:** Ask the server to return a diagnostic trace of the actions it takes.
- **OPTIONS:** Ask the server to return the list of request methods it supports.
- **CONNECT:** Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.

HTTP response status codes

- The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.
- HTTP response status codes **are three-digit numbers returned by a web server to indicate the outcome of an HTTP request** made by a client (such as a web browser or API client).
- HTTP status codes are grouped into five classes, each with a specific meaning:
- **1xx (Informational):**
  - These status codes indicate that the server has received the initial part of the request and is continuing to process it. These codes are typically informational and not commonly used in practice.

- **2xx (Success):**
- These status codes indicate that the request was successfully received, understood, and processed by the server.
- Common 2xx codes:
  - 200 (OK): The request was successful, and the server is returning the requested data.
  - 201 (Created): The request resulted in the creation of a new resource on the server.
  - 204 (No Content): The request was successful, but there is no data to return (often used for DELETE requests).

- **3xx (Redirection):**
- These status codes indicate that further action is needed by the client to complete the request. They are typically used for URL redirection and provide information on where to find the requested resource.
- Common 3xx codes:
  - 301 (Moved Permanently): The requested resource has permanently moved to a new URL.
  - 302 (Found): The requested resource is temporarily located at a different URL (often used for temporary redirects).
  - 304 (Not Modified): The client's cached copy of the resource is still valid; no new data is sent.

4xx (Client Error):

- The request contains bad syntax or cannot be understood.
- These status codes indicate that the client has made an error in the request, such as providing invalid data or attempting to access a resource without proper authorization.
- Common 4xx codes:
  - 400 (Bad Request): The request is malformed or contains invalid data.
  - 401 (Unauthorized): Authentication is required, and the client's credentials are missing or invalid.
  - 403 (Forbidden): The client is authenticated but does not have permission to access the requested resource.
  - 404 (Not Found): The requested resource does not exist on the server.
  - 429 (Too Many Requests): The client has exceeded its rate limit for making requests.

5xx (Server Error):

- These status codes indicate that the server encountered an error while processing the request and was unable to fulfill it.
- Common 5xx codes:
  - 500 (Internal Server Error): A generic error message indicating that something went wrong on the server's end.
  - 502 (Bad Gateway): The server, acting as a gateway or proxy, received an invalid response from the upstream server.
  - 503 (Service Unavailable): The server is temporarily unable to handle the request due to maintenance or overload.
  - 504 (Gateway Timeout): The server, acting as a gateway or proxy, did not receive a timely response from the upstream server.

How do we access/test out these HTTP requests

- There are several ways to test HTTP requests, depending on your specific needs and the tools or technologies you prefer.
- The most common methods for testing HTTP requests include:

1. Web Browsers:

- You can test GET requests by simply entering a URL in your web browser's address bar and pressing Enter.
- You can use browser extensions like Postman or RESTClient to send various HTTP requests (GET, POST, PUT, DELETE) and inspect theresponses.
- Modern web browsers come with **browser developer tools** that include a Network tab, which allows you to monitor HTTP requests and responses made by a web page. You can use this for debugging and inspection.

2. API Testing Tools:

- Postman: Postman is a popular API testing tool that provides a graphical user interface for sending and testing HTTP requests. It allows you to organize and save requests, add headers and parameters, and view responses.
- Insomnia: Insomnia is another API testing tool that offers a user-friendly interface for creating and testing HTTP requests. It also supports features like environment variables and code generation.
- Automated Testing Frameworks:
  - if you're testing APIs as part of automated testing, you can use testing frameworks like Selenium (for web applications), REST-assured (for REST APIs in Java), or similar tools in other programming languages.

3. Programming Languages:

- You can write code in various programming languages to send HTTP requests.
- Popular libraries and frameworks like requests in Python, axios in JavaScript, or HttpClient in C# can be used to craft and send requests programmatically.
- Custom Scripts:
  - You can write custom scripts or code to send HTTP requests for specific testing scenarios using the programming language of your choice.

4. Command-Line Tools:

- **cURL:** This is a command-line tool that allows you to make HTTP requests from the terminal. You can use it to send GET, POST, PUT, DELETE, and other types of requests.
- **HTTPie**: Similar to cURL, HTTPie is a user-friendly command-line HTTP client that makes it easy to send HTTP requests and view responses.

- The choice of method depends on your specific testing needs, your familiarity with the tools, and the complexity of the tests you want to perform

- Utility programs like "telnet" and "HyperTerminal" can be used for testing network connectivity and manually crafting and sending raw HTTP requests.
- But they are not commonly used for modern HTTP request testing due to their limitations and security concerns.
- **Telnet:**
  - Telnet is a command-line utility that allows you to establish a basic, unencrypted text-based terminal connection to a remote server over a network. It can be used to manually send HTTP requests by typing out the HTTP headers and content.
  - Telnet can be used for basic testing but, it lacks the features and convenience of modern HTTP testing tools. It doesn't provide a user-friendly interface for managing headers, request bodies, or viewing responses.
  - Importantly, Telnet does not support secure (HTTPS) connections, making it unsuitable for testing encrypted connections, which are essential for most web applications today.

Curl

- Client URL (cURL, pronounced “curl”) is a command line tool that enables data exchange between a device and a server through a terminal.
- Using the command line interface (CLI), a user specifies a server URL (the location where they want to send a request) and the data they want to send to that server URL.
- API tools like Postman and Insomnia provide an interactive user interface (UI) that allows you to make different forms of requests using URLs, for receiving and processing requests.
- The cURL command does the same thing, except in your terminal. cURL works on Linux, Mac, and Windows.
- The cURL command uses the libcURL client-side URL transfer library. This library supports many different transfer protocols including HTTPS, SMTP, and FTP.

How to use curl

- Run the **curl command** from the command line with specific options and arguments.
- curl [options] [URL]
- General Options:
  - -X: Set the HTTP request method.
  - -H: Add custom headers to the request.
  - --data or -d: Send data in the request body (used with POST requests).
  - -i: Include HTTP response headers in the output.
  - -o: Save the response body to a file.
  - -O: Save the response body using the remote file name.
  - -w: Specify a custom format for the output.
  - -u: Provide basic authentication credentials.
  - --url: Set the URL for the request.
  - --cookie: Send a cookie in the request header.
  - --cookie-jar: Store cookies in a file.
  - -v: Enable verbose output for debugging.
  - --trace: Log detailed information about the request and response to a file.

COOKIES

- HTTP cookies (also called web cookies, Internet cookies, browser cookies, or simply cookies) are small blocks of data created by a web server while a user is browsing a website and placed on the user's computer by the user's web browser.
- Cookies are placed on the device used to access a website, and more than one cookie may be placed on a user's device during a session.
- They are commonly used for maintaining state and tracking user information as they interact with a website.

Purpose of Cookies

- Session Management:
  - Cookies are often used to manage user sessions on websites. When a user logs in, a **session cookie** is created to identify the user throughout their visit. This allows the server to maintain information about the user's session, such as their authentication status and shopping cart contents.
- User Authentication:
  - Cookies are used to authenticate users. A server can issue a session cookie upon successful login, and subsequent requests include this cookie to verify the user's identity without requiring them to log in repeatedly.
- Personalization:
  - Cookies can store user preferences and settings, enabling a website to provide a customized experience. For example, a site can remember a user's language preference or theme choice.
- Tracking and Analytics:
  - Cookies are used for tracking user behavior and gathering analytics data. This information helps website owners understand how users interact with their site, which can inform improvements and marketing strategies.
- Targeted Advertising:
  - Advertisers use cookies to track users' interests and behavior online. This data is used to deliver targeted ads to users based on their browsing habits.