**ASSIGNMENT**

1. What are the basic Features of HTTP?

The features of HTTP are

**Connectionless**:

The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back after which client disconnect the connection. So client and server knows about each other during current request and response only.

**Media Independent:**

 It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.

**Stateless:**

As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

1. What are request methods in HTTP?

The request methods in HTTP are

**GET** : is used to request data from specified resource.

**POST**  is used to send data to the server create/update a resource.

**PUT is used to send data to a server to create/update a resource.**

**Delete is used to delete the resource.**

1. What are the differences between GET and POST methods?
   1. GET requests are re-executed but may not be re-submitted to server if the HTML is stored in the browser cache.

**POST** browser usually alerts the user data will need to resubmitted.

* 1. GET: can send but the parameter data is limited to what we can stuff into the request line (URL). Safest to use less than 2K of parameters, some servers handle up to 64K.

POST: Can send parameters, including uploading files, to the server.

* 1. GET is less secure compared to POST because data sent is part of the URL. So it's saved in browser history and server logs in plaintext.

POST is a little safer than GET because the parameters are not stored in browser history or in [web server](https://www.diffen.com/difference/Application_Server_vs_Web_Server) logs.

* 1. GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

POST method variables are not displayed in the URL.

1. What is status code in HTTP?

Status codes are issued by a server in response to a [client's request](https://en.wikipedia.org/wiki/Client_(computing)) made to the server. It includes codes from IETF [Request for Comments](https://en.wikipedia.org/wiki/Request_for_Comments) (RFCs), other specifications, and some additional codes used in some common applications of the HTTP. The first digit of the status code specifies one of five standard classes of responses.

Responses are grouped into five classes:

1. Informational responses (100–199),
2. Successful responses (200–299),
3. Redirects (300–399),
4. Client errors (400–499),
5. Server errors (500–599).
6. What are the header fields in HTTP?

The header fields in HTTP are

**General-header:** These header fields have general applicability for both request and response messages.

**Client Request-header:** These header fields have applicability only for request messages.

**Server Response-header:** These header fields have applicability only for response messages.

**Entity-header:** These header fields define meta information about the entity-body or, if no body is present, about the resource identified by the request.

1. What is URI?

A Uniform Resource Identifier (URI) is a string of [characters](https://en.wikipedia.org/wiki/Character_(computing)) that unambiguously [identifies](https://en.wikipedia.org/wiki/Identifier) a particular [resource](https://en.wikipedia.org/wiki/Web_resource). To guarantee [uniformity](https://en.wiktionary.org/wiki/uniformity), all URIs follow a predefined set of syntax rules, but also maintain [extensibility](https://en.wikipedia.org/wiki/Extensibility) through a separately defined [hierarchical](https://en.wikipedia.org/wiki/Hierarchical) naming scheme (e.g. http://).

Such identification enables interaction with representations of the resource over a network, typically the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web), using specific [protocols](https://en.wikipedia.org/wiki/Communications_protocol). Schemes specifying a concrete [syntax](https://en.wikipedia.org/wiki/Syntax) and associated protocols define each URI. The most common form of URI is the Uniform Resource Locator ([URL](https://en.wikipedia.org/wiki/URL)), frequently referred to informally as a web address. More rarely seen in usage is the [Uniform Resource Name](https://en.wikipedia.org/wiki/Uniform_Resource_Name) (URN), which was designed to complement URLs by providing a mechanism for the identification of resources in particular [namespaces](https://en.wikipedia.org/wiki/Namespace).

1. What are Idempotent methods and why do we call them?

An idempotent method is a method that can be called many times without different outcomes. It would not matter if the method is called only once or ten times over. Again this only applies to the result, not the resource itself. This still can be manipulated (like an update timestamp, provided this information is not shared in the (current) resource representation.

GET, PUT, DELETE, HEAD and TRACE are idempotent.

When you design REST APIs, you must realize that API consumers can make mistakes. They can write client code in such a way that there can be duplicate requests as well. These duplicate requests may be unintentional as well as intentional some time (e.g. due to timeout or network issues). You have to design fault-tolerant APIs in such a way that duplicate requests do not leave the system unstable.

1. Explain HTTP Request & Response Messages

HTTP messages are how data is exchanged between a server and a client. There are two type of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server.

HTTP messages are composed of textual information encoded in ASCII, and span over multiple lines. In HTTP/1.1, and earlier versions of the protocol, these messages were openly sent across the connection. In HTTP/2, the once human-readable message is now divided up into HTTP frames, providing optimization and performance improvements.

Web developers, or webmasters, rarely craft these textual HTTP messages themselves: software, a Web browser, proxy, or Web server, perform this action. They provide HTTP messages through config files (for proxies or servers), APIs (for browsers), or other interfaces.

1. What is Session State in HTTP?

Session state, is a method keep track of the a user session during a series of HTTP requests. Session state allows a developer to store data about a user as he/she navigates through ASP.NET web pages in a .NET web application.

The concept of a session is generic and applies to most web servers. Session state however is a Microsoft-centric concept. Sessions are started during the first request and session values will persist as long as a new request is made by the browser before the number of minutes specified in the [Timeout](https://docs.microsoft.com/en-us/dotnet/api/system.web.sessionstate.httpsessionstate.timeout?view=netframework-4.8) property pass. When a new session begins, the session [Start](https://docs.microsoft.com/en-us/dotnet/api/system.web.sessionstate.sessionstatemodule.start?view=netframework-4.8) event is raised. You can use this event to perform any additional work at the start of a session, such as setting default session values.

1. What is HTTPs?

Hypertext Transfer Protocol Secure (HTTPS) is an extension of the [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP). It is used for [secure communication](https://en.wikipedia.org/wiki/Secure_communications) over a [computer network](https://en.wikipedia.org/wiki/Network_operating_system), and is widely used on the Internet. In HTTPS, the [communication protocol](https://en.wikipedia.org/wiki/Communication_protocol) is encrypted using [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security) (TLS) or, formerly, Secure Sockets Layer (SSL). The protocol is therefore also referred to as HTTP over TLS, or HTTP over SSL.

The principal motivations for HTTPS are [authentication](https://en.wikipedia.org/wiki/Authentication) of the accessed [website](https://en.wikipedia.org/wiki/Website), and protection of the [privacy](https://en.wikipedia.org/wiki/Information_privacy) and [integrity](https://en.wikipedia.org/wiki/Data_integrity) of the exchanged data while in transit. It protects against [man-in-the-middle attacks](https://en.wikipedia.org/wiki/Man-in-the-middle_attack), and the bidirectional [encryption](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation) of communications between a client and server protects the communications against [eavesdropping](https://en.wikipedia.org/wiki/Eavesdropping) and [tampering](https://en.wikipedia.org/wiki/Tamper-evident#Tampering). In practice, this provides a reasonable assurance that one is communicating with the intended website without interference from attackers.