# QUESTION 1

#Overview of HTTP, Requests and Responses

HTTP, or HyperText Transfer Protocol, is a protocol (a set of rules) in the Internet Protocol suite (IP) for fetching or posting resources, such as HTML documents. HTTP uses a client-server model to exchange information. The client is commonly a user agent such as a web browser, whereas the server is a web server which stores the HTML document. HTTP is built within the Internet protocol suite, and assumes the presence of an underlying transport layer protocol. TCP is commonly used for this purpose.

The exchange of information between client and server happens as follows:

1) The client (web browser) opens a TCP connection with the server.
2) The client sends an HTTP Request to the server.
3) The server reads the Request and sends a Response to the client, which may contain the requested resources.
4) The connection is closed or another request is sent.

A little more about Requests and Responses:

Requests, sent by the client, have a format which consists of the request type, called method (usually GET/POST), then the path of the requested resource, that is, the URL, without elements which are obvious such as the protocol, TCP port or domain name. The version of the HTTP protocol is also mentioned, and optional headers are present. In some methods like POST, a body is also present similar to that in the Response.

The Response also has a structure, and contains the HTTP protocol version, the status code and message, HTTP headers and optionally, the document fetched in the body.

It should be noted that there are various proxy servers present between client and server, and also routers and modems, which perform various functions such as clearing cache, routing the message and response signals, amplifying signal strength, etc.

HTTP is also “stateless”, meaning every command is carried out independent of others.

#HTTP methods in a request

The common HTTP methods in requests are POST, GET, PUT, PATCH and DELETE. These correspond to the Create, Read, Update and Delete operations (CRUD). Apart from these, HEAD, CONNECT, TRACE and OPTIONS are also present.

1) GET: Used to retrieve resources from the server. Returns status message “200 OK” if resource is found, and “404 NOT FOUND” if not. Use-case could be requesting a particular page on Wikipedia for reference.

2) POST: Used to submit data. Has use-cases such as the submission of a form, or creating an account on a website by filling the sign up info, or logging in to the same site.
3) PATCH: Used to update the website. Can be used to, for example, update your account credentials on LinkedIn.
4) DELETE: Used to delete the requested resource. For example, if you wish to delete your account.
5) PUT: PUT works like UPDATE if the requested resource already exists, and updates it. - However, if it does not exist, PUT will create it. Can be used for sign-up.

HEAD is similar to GET but does not ask for a response body.

Some of these methods are idempotent, meaning the result would be the same no matter how many times we call them. Some are "safe" too, such as GET, meaning the request does not modify the contents of the resource.

#User-Agent header

Requests and Responses can be viewed by right-clicking on the webpage, selecting "Inspect" which opens Dev Tools, and choosing the "Networks" tab. The request names can be seen on the left and Headers can be viewed by selecting the Headers tab on the right.

In the context of HTTP, the User-Agent is any software which sends HTTP requests on behalf of the user. This is usually the web browser. The User-Agent header of an HTTP request, thus, is a string which allows servers to identify the specifics of the User-Agent, typically the browser.

#More on the User-Agent string

The history of the User-Agent string is very interesting. In the early-days of the Internet, we had a browser called NCSA Mosaic. Then, Mozilla was developed. Mozilla was short-hand for "Mosaic killer", but Mosaic objected to this and it was renamed Netscape. But in its User-Agent the name remained Mozilla. Mozilla supported frames- dividing the browser-window into sub-sections, a feature which was loved by users, but Mosaic did not support frames. So "user-agent sniffing" started so that servers could send pages with frames to Mozilla but not Mosaic.

Now, Microsoft developed Internet Explorer, which supported frames too. But servers had been programmed to send frames only to Netscape, and Microsoft, not wishing to wait for a reprogram, spoofed Mozilla- pretended to be Mozilla in its user-agent string. This way servers sent it frames too.

IE was better than Netscape, and overtook it. But Mozilla built Gecko, a rendering engine. The Firefox browser of Mozilla used Gecko. Gecko called itself *Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.1) Gecko/20020826*, and many other browsers used Gecko, all having Mozilla/Gecko in their User-Agent. Gecko was more advanced than other engines of that time.

User-sniffing started again and browsers running on Gecko were given better pages than others.

Linux users used Konqueror, running on KHTML, which was as advanced as Gecko but still wasn't sent good pages as it wasn't Gecko. So Konqueror spoofed Gecko in its User-Agent.

Apple built Safari, and used KHTML, but added many features, and forked the project, and called it WebKit, but wanted pages written for KHTML, and so Safari called itself *Mozilla/5.0 (Macintosh; U; PPC Mac OS X; de-de) AppleWebKit/85.7 (KHTML, like Gecko) Safari/85.5.*

And then Google built Chrome, and Chrome used Webkit, and it was like Safari, and wanted pages built for Safari, and so pretended to be Safari. And thus Chrome pretended to be Safari, and WebKit pretended to be KHTML, and KHTML pretended to be Gecko, and all browsers pretended to be Mozilla.

Even though Mozilla is a company now, in this context it refers to the old browser which replaced Mosaic.

# HTTP Headers

The User-Agent header has been described extensively.

Here are a few more:

1) Authorization: This is a token which tells the server whether the user is allowed to access the resources requested.
2) Content-Type: In responses, this tells the client the type of data in the request body, i.e., the data type which the user expects to receive.
3) Host: This request header specifies the host and port number of the server to which the request is being sent.

Citations:


- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
- https://doc.oroinc.com/api/http-methods/#:~:text=The%20primary%20or%20most%20commonly,they%20are%20utilized%20less%20frequently.
- https://stackoverflow.com/questions/5125438/why-do-chrome-and-ie-put-mozilla-5-0-in-the-user-agent-they-send-to-the-server
- https://stackoverflow.com/questions/1114254/why-do-all-browsers-user-agents-start-with-mozilla
- https://webaim.org/blog/user-agent-string-history/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent
- https://alexzitowolf.medium.com/the-5-most-important-http-headers-d9e9f94bb1f6
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Host


# QUESTION 2

# What Cookies are, How they are set, and Why

Cookies are text files containing small pieces of data which can be used to identify a particular computer in a network. Specifically, web cookies are used to personalize websites for users and enhance the user experience.

Cookies are mainly of two types, magic cookies and the currently popular HTTP cookies. HTTP cookies are an improvement upon magic cookies. Let us describe HTTP cookies in a bit more detail. HTTP cookies are built for Internet web browsers to track, personalize, and save information about each user's session.

When you visit a website, the server sends a stream of identifying data to the user-agent, like a ticket. This consists of "name-value" pairs (cookie crumbs). This is the cookie, which the web browser stores locally. When the user visits the site again, the browser sends this cookie to the server which accordingly personalizes the response.

Cookies are set using the Set-Cookie header field in the HTTP response. When the user visits a site for the first time, the server sends an HTTP response to the Request containing "Set-Cookie" header fields telling the browser to store particular cookies for return visits. Upon a return visit, the web browser sends a Request with the "Cookie" header containing the cookie.

There are various reasons to set cookies. These include:

1) Cookies allow websites to recognize users and recall their preferences and login information.
2) Cookies are used to send targeted, personalized advertisements to the user on the basis of their browsing habits on a website.
3) Sites such as online stores and food delivery sites use cookies to store items which the user previously liked or bought, so that similar items can be further recommended.
4) Cookies are also used for authentication.

# Cookie Attributes

1) Domain: This attribute is used to add the domain for which the cookie is valid. For instance, if the domain value is "example.com", then the cookie will be valid for example.com and its subdomains.
2) Path: The path attribute is used to control the scope of cookies across webpages in a domain. This can be used to restrict cookies for certain pages.
3) Expires: Defines a date and time when the browser should delete the cookie.
4) Max-Age: This attribute also defines the lifetime of a cookie, but relative to the time of creation of the cookie rather than as an absolute date and time. Max-Age defines the age of the cookie in seconds in the future, after which time it will be deleted.
5) Secure: Directs browsers to use cookies only via encrypted connections.
6) HttpOnly: This attribute directs browsers not to expose cookies through channels other than HTTP (and HTTPS) requests. This means that the cookie cannot be accessed via client-side scripting languages.

# How cookie help companies track users across sites

Advertising companies such as Google, Facebook, Twitter, etc use Third-Party Persistent Cookies (Tracking cookies) to track users around the web. Third-Party cookies appear on websites but aren't created by the website owner. Thus, these cookies have a Domain attribute different from the requested URL. These cookies are also persistent, meaning they remain stored on the browser after

you close the site. These cookies come when a website owner uses resources belonging to other sites.

Tracking cookies are usually used for advertising purposes, **retargeting** in particular. Retargeting is a marketing tactic involving showing users ads who have previously shown interest in a specific product.
So, what an advertising company does is to set third party cookies in various different sites, so that information about the user can be tracked from all the different sites which the user visits and which has third-party cookies from the advertiser.

# FLoC
The privacy concerns from cookies are obvious. Tracking cookies from large advertising companies are almost ubiquitous on the Internet. So, these companies can track users across almost all sites they visit, which means they can put together a sort of browsing history for the user. Moreover, they can track search queries, purchases, device information, location, when and where you saw previous advertisements, how many times you've seen an ad, and what links you click on.
All this can be done without the knowledge or consent of the user.

To solve this issue, Google proposed an alternative to cookies called FLoC (Federated Learning of Cohorts).
Under this system, instead of individual tracking, users will be grouped into cohorts based on their browsing habits, and each cohort will have a few thousand users with similar habits, who can then be shown interest-based ads. Cohorts would also be updated weekly. This would make tracking individual users more difficult.
However, this system was heavily criticised. In the third-party tracking system, each website interested in tracking had to do so individually. But under the FLoC system, the browser itself would do the tracking for them. This meant that every website which opts into the FLoC system would have information about a user's browsing habits without even having to track them. This was also labelled anti-competitive, since now Chrome would act as a "gatekeeper" of tracking users.

# Citations

- https://www.kaspersky.com/resource-center/definitions/cookies
- https://en.wikipedia.org/wiki/HTTP_cookie#:~:text=Cookies%20are%20set%20using%20the,cookies%20or%20has%20disabled%20cookies).
- https://linuxhint.com/javascript-cookies-attributes/#:~:text=%E2%80%9Cexpires%E2%80%9D%2C%20%E2%80%9Cmax%2D,and%20the%20lifetime%20of%20cookies.
- https://privacy.net/stop-cookies-tracking/
- https://en.wikipedia.org/wiki/Federated_Learning_of_Cohorts


# QUESTION 3

# When is the CORS mechanism required?

The CORS mechanism is required to enable cross-sharing of resources, that is, this mechanism allows a web browser to obtain resources from a domain other than the requested domain.
This is necessary in certain cases, and CORS allows the free sharing of cross-origin images, stylesheets, scripts, iframes, and videos.

CORS was implemented due to the restrictions revolving around the same-origin policy. This policy limited certain resources to interact only with resources from the parent domain. This was necessary to enhance security.
However, in some cases, it is quite beneficial to enable Cross-Origin Resource Sharing as it allows for additional freedom and functionality for websites. This is true in many cases these days for web fonts and icons which are often requested from another domain. In this case, with the use of HTTP headers, CORS enables the browser to manage cross-domain content by either **allowing or denying** it based on the configured security settings.

# Headers in CORS

CORS Headers in Requests are as follows:

1) Origin: Indicates the domain of origin of the CORS request. The Host attribute of the HTTP request is the server from which the "Origin" server requests resources.
2) Access-Control-Request-Header: This is sent in preflighted-requests, telling the server what HTTP headers will be used when the actual request is made.
3) Access-Control-Request-Method: This is sent in preflighted-requests, telling the server what HTTP method will be used when the actual request is made.

CORS Headers in Responses are as follows:

1) Access-Control-Allow-Origin: Tells the browser to allow the Origin to access the resource from Host. The * wildcard allows any origin to access resources.
2) Access-Control-Expose-Headers: Names the response Headers which the browser is allowed to access.
3) Access-Control-Max-Age: Indicates how long the results of a preflight-request can be cached/stored.
4) Access-Control-Allow-Methods: Specifies methods allowed when accessing the resource. Used as a response to a preflight request.
5) Access-Control-Allow-Credentials: Indicates whether response to request can be exposed if credentials flag is true. Used as part of a response to a preflight request, this indicates whether or not the actual request can be made using credentials.
6) Access-Control-Allow-Headers: Used in a preflight request, this indicates the headers that can be used in the actual request.

# CORS preflight requests

Cross-origin requests which have implications for user data, that is, requests which can change user data (Eg: HTTP methods apart from GET) have to be preflighted. This means an additional request is sent before the actual request to determine whether permission to perform the actual request is present.
These requests are made with the OPTIONS method. A simple example of a preflight request, sourced from Wikipedia, is shown below:

```
OPTIONS /
Host: service.example.com
Origin: http://www.example.com
Access-Control-Request-Method: PUT
```

# Citations


- https://en.wikipedia.org/wiki/Cross-origin_resource_sharing
- https://www.keycdn.com/support/cors
- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#what_requests_use_cors
- https://www.ibm.com/docs/en/sva/10.0.1?topic=control-cross-origin-resource-sharing-cors-policies


# QUESTION 4

# JWT structure and basic working

JWT stands for JSON Web Token. JSON stands for JavaScript Object Notation, and is a text format for storing and transporting data. It stores data in key-value pairs.
JWTs are used for session management, so that a user can safely remain logged in a website. JWTs are an alternative to the traditional sessions cookie approach.

In this approach, the server generates a JWT-an encrypted token, which it sends to the browser. The browser stores the token locally. This token itself contains information about the user. So, unlike the cookie approach where a SessionID has to be stored in the database in the server-end, no data about the user is stored on the server-end in the JWT approach. Instead, the server checks the token, which itself guarantees the authenticity of the user.

The structure of the JWT is as follows:
In the encrypted form, it consists of three concatenated Base64url-encoded strings, separated by dots. In the decrypted form, it has 3 parts- header, payload and signature.

- **JOSE Header**: contains metadata about the type of token and the cryptographic algorithms used to secure its contents.
- **JWS payload**: contains verifiable security statements, such as the identity of the user and the permissions they are allowed.
- **JWS signature**: used to validate that the token is trustworthy and has not been tampered with.

# Merits and Demerits

The major advantage of using a JWT instead of Session Cookie is that there is no need of a server-end database storing session Ids. This becomes a problem in large websites with hundreds of millions of users. So database lookups aren't needed. JWTs also prevent cross-site scripting attacks.

On the other hand, JWTs are created by cryptographic algorithms which tend to be slow. Thus the process of JWT creation is slower than cookie creation.

Sometimes additional and unnecessary information is stored in the JWT. The JWT token should primarily contain user information, and the data authorized to be accessed by that user should be

provisioned and managed as a separate service on that respective server. Tokens also get invalidated after a particular amount of time, so refresh tokens have to be used to prevent this.

#Confidentiality and Integrity in JWT

A JWT ensures data integrity using the signature section, but confidentiality is not assured and information leaks are possible albeit highly unlikely.

JWTs can be either signed, encrypted or both. If a token is signed, but not encrypted, everyone can read its contents, but without the private key, you can't change it. Otherwise, the receiver will notice that the signature won't match anymore. So the data remains secure but not confidential in non-encrypted JWTs.

#CASI

I found the JWT token by going to the Applications tab of DevTools, and under Storage>Cookies, the “token” was present in encoded form.

Then I decoded it using https://developer.pingidentity.com/en/tools/jwt-decoder.html

> Upon navigating to Citadel the token was still present. I believe Citadel was able to access the token because of an intermediate domain between CASI and Citadel. To authenticate in CASI, the user is pointed to the authentication server in an intermediate domain, where a JWT is issued and stored in the browser, then the user is redirected to the origin domain CASI. Basically, a central authentication server is present for both Citadel and CASI.

# Citations


- https://blog.miniorange.com/what-is-jwt-json-web-token-how-does-jwt-authentication-work/#:~:text=JWT%20authentication%20is%20a%20token,they%20are%20typically%20encoded%20%26%20signed.
- https://medium.com/@prashantramnyc/difference-between-session-cookies-vs-jwt-json-web-tokens-for-session-management-4be67d2f066e#:~:text=The%20JWT%20tokens%20are%20sometimes,by%20the%20%E2%80%9Csecret%20key%E2%80%9D.
- https://auth0.com/docs/secure/tokens/json-web-tokens/json-web-token-structure
- https://fusionauth.io/learn/expert-advice/tokens/pros-and-cons-of-jwts
- https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a
- https://stackoverflow.com/questions/27301557/if-you-can-decode-jwt-how-are-they-secure
- https://stackoverflow.com/questions/33723033/single-sign-on-flow-using-jwt-for-cross-domain-authentication

JWT Pictures: