Name: Harshit Mawandia

Research Project for DevClub

HTTP:

The **Hypertext Transfer Protocol** (**HTTP**) is a layer of rules that allows communication between servers and clients. It is the foundation for data communication for the World wide web. It functions as a request-response protocol in the client-server computing model.

It exchanges messages between the client and the server via requests and responses. The client submits an HTTP *request* message to the server. The server, which provides *resources* such as HTML files and other content, or performs other functions on behalf of the client, returns a *response* message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

HTTP requests are messages sent by the client to initiate an action on the server. Their *start-line* contains three elements:

1. An *HTTP method*, a verb (like `GET`, `PUT` , or `POST`) or a noun (like `HEAD` or `OPTIONS`) that describes the action to be performed
2. The *request target*, usually a URL, or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods.
3. The *HTTP version*, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

Different Methods that can be specified in a request are:

- GET **Method**
- HEAD **Method**
- POST **Method**
- PUT **Method**
- DELETE **Method**
- CONNECT **Method**
- OPTIONS **Method**
- TRACE **Method**

GET method example:

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

POST method example:

POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: 88
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

PUT method example:

PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

How to see the request and responses being sent by your web browser?

To see the requests and responses being sent by a browser, head to the network tab in the developer console; you'll see the name of the website, click on it, and head to the header section to see the requests and responses sent and received.

The user agent for google.com is:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90 .0.4430.85 Safari/537.36

Gecko/geckotrail indicates that the browser is based on Gecko. (On Desktop, ***geckotrail*** is always the fixed string 20100101.)

What does Mozilla mean in this context?

Mozilla/5.0: Previously used to indicate compatibility with the Mozilla rendering engine.

What happened in summary:

1. Mozilla browser gets released, with User-Agent Mozilla/1.0 (Win3.1). It is publicly renamed to Netscape, but in its User-Agent, it keeps its original name.
2. Internet Explorer is released. It spoofs Netscape by starting its User-Agent with Mozilla/ because web servers were routinely browser sniffing and serving pages with frames - a feature supported by both Netscape and IE, but not other browsers of the era - to Netscape only.
3. Over time, Gecko, Konqueror, Opera, Safari and Chrome each decide to similarly spoof the User-Agent of some previous browser in order to manipulate browser-sniffing web pages into correctly understanding their browser's features. As part of this spoofing, all the browsers start their User-Agents with Mozilla/, like the browsers before them had done. Plenty of other nonsense also results, like modern Chrome's User-Agent simultaneously claiming to be Mozilla, Chrome, Safari, and 'like Gecko'.

HTTP headers:

HTTP headers let the client and the server pass additional information with an HTTP request or response. They define the operating parameters of an HTTP transaction.

User Agent:

The User-Agent request header is a characteristic string that lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent.

Host header:

The Host request header specifies the host and port number of the server to which the request is being sent. It has he domain name of the server (for virtual hosting), and the TCP port number on which the server is listening. The port number may be omitted if the port is the standard port for the service requested.

Cookies:

HTTP cookies**,** or internet cookies, are built specifically for Internet web browsers to track, personalize, and save information about each user's session. A "session" just refers to the time you spend on a site.

Cookies are created to identify you when you visit a new website. The web server — which stores the website's data — sends a short stream of identifying info to your web browser.

If a user returns to that site in the future, the web browser returns that data to the web server in the form of a cookie. This is when your browser will send it back to the server to recall data from your previous sessions.

Some of the different cookie attributes are:

- Secure Attribute. The Secure attribute tells the browser to only send the cookie if the request is being sent over a secure channel such as HTTPS.
- HttpOnly Attribute. The HttpOnly attribute is used to help prevent attacks such as session leakage, since it does not allow the cookie to be accessed via a client-side script such as JavaScript.
- Domain Attribute. The Domain attribute is used to compare the cookie's domain against the domain of the server for which the HTTP request is being made. If the domain matches or if it is a subdomain, then the path attribute will be checked next.
- Path Attribute. The Path attribute plays a major role in setting the scope of the cookies in conjunction with the domain. In addition to the domain, the URL path that the cookie is valid for can be specified. If the domain and path match, then the cookie will be sent in the request
- Expires Attribute.

  The Expires attribute is used to:

  - set persistent cookies
  - limit lifespan if a session lives for too long
  - remove a cookie forcefully by setting it to a past date

- SameSite Attribute. The SameSite attribute is used to assert that a cookie ought not to be sent along with cross-site requests

Cookies are used for:

1. **Session management.** For example, cookies let websites recognize users and recall their individual login information and preferences, such as sports news versus politics.
2. **Personalization.** Customized advertising is the main way cookies are used to personalize your sessions. You may view certain items or parts of a site, and cookies use this data to help build targeted ads that you might enjoy.
3. **Tracking.** Shopping sites use cookies to track items users previously viewed, allowing the sites to suggest other goods they might like and keep items in shopping carts while they continue shopping.

How cookies help in advertisements:

Cookies collect information about your Internet habits: the pages you visit frequently and the topics that interest you. The problem is that they usually share this information with data analysis firms or those that design targeted marketing campaigns.
If, say, an ad for a food product appears on your screen after you visit a restaurant page, don't be too surprised. Thanks to cookies, advertising can be tailored to consumers' preferences.

What are the privacy concerns with cookies?

Since tracking cookies are used to gather information about you without your authorization, they present a real threat to your online privacy. Tracking cookies like third-party cookies aren't used to enhance your experience but rather to keep track of your activity across certain websites.

All this information can be used to create browsing history profiles, so you can be targeted with specific ads.

Also, did you know that malware and viruses can be disguised as Internet cookies? Supercookies can be sent by a hacker in control of a malicious website to potentially impersonate or interrupt legitimate user requests to another website with a similar public suffix or top-level domain.

This allows the hacker to fake login into the website as you and use your personal information for their nefarious purposes.

You may feel like someone is watching you as you're browsing the internet, which understandably makes some feel uncomfortable

Federated Learning of Cohorts (FLoC)

FLoC provides a privacy-preserving mechanism for interest-based ad selection.Federated Learning of Cohorts (FLoC) proposes a new way for businesses to reach people with relevant content and ads by clustering large groups of people with similar interests. This approach effectively hides individuals "in the crowd" and uses on-device processing to keep a person's web history private on the browser.

Criticism of FLoC:

Federated learning requires frequent communication between nodes during the learning process. Thus, it requires not only enough local computing power and memory, but also high bandwidth connections to be able to exchange parameters of the machine learning model. However, the technology also avoid data communication, which can require significant resources before starting centralized machine learning. Nevertheless, the devices typically employed in federated learning are communication-constrained, for example IoT devices or smartphones are generally connected to Wi-fi networks, thus, even if the models are commonly less expensive to be transmitted compared to raw data, federated learning mechanisms may not be suitable in their general form.

Federated learning raises several statistical challenges:

- Heterogeneity between the different local datasets: each node may have some bias with respect to the general population, and the size of the datasets may vary significantly;
- Temporal heterogeneity: each local dataset's distribution may vary with time;
- Interoperability of each node's dataset is a prerequisite;

- Each node's dataset may require regular curations;
- Hiding training data might allow attackers to inject backdoors into the global model;
- Lack of access to global training data makes it harder to identify unwanted biases entering the training e.g. age, gender, sexual orientation;
- Partial or total loss of model updates due to node failures affecting the global model.

Cross-origin resource sharing (CORS):

CORS is needed when restricted resources on a web page are to be requested from another domain outside the domain from which the first resource was served. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos. Certain "cross-domain" requests, notably Ajax requests, are forbidden by default by the same-origin security policy. CORS defines a way in which a browser and server can interact to determine whether it is safe to allow the cross-origin request. It allows for more freedom and functionality than purely same-origin requests, but is more secure than simply allowing all cross-origin requests.

The HTTP headers that relate to CORS are:

Request Headers

- Origin
- Access-Control-Request-Method
- Access-Control-Request-Headers

Response headers[edit]

- Access-Control-Allow-Origin
- Access-Control-Allow-Credentials
- Access-Control-Expose-Headers
- Access-Control-Max-Age
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers

Their values can be : *, <origin> and null

*

For requests *without credentials*, the literal value "*" can be specified, as a wildcard; the value tells browsers to allow requesting code from any origin to access the resource. Attempting to use the wildcard with credentials will result in an error.

<origin>

Specifies an origin. Only a single origin can be specified. If the server supports clients from multiple origins, it must return the origin for the specific client making the request.

null

Specifies the origin "null".

What are CORS preflight requests?

A CORS preflight request is a CORS request that checks to see if the CORS protocol is understood and a server is aware using specific methods and headers.

It is an `OPTIONS` request, using three HTTP request headers: `Access-Control-Request-Method`, `Access-Control-Request-Headers`, and the `Origin` header.

A preflight request is automatically issued by a browser and in normal cases, front-end developers don't need to craft such requests themselves. It appears when request is qualified as "to be preflighted" and omitted for simple requests.

For example, a client might be asking a server if it would allow a `DELETE` request, before sending a `DELETE` request, by using a preflight request:

CORS preflight request has:

1. The request's HTTP method is OPTIONS
2. It has an Origin header
3. It has an Access-Control-Request-Method header, indicating what's the actual method it's trying to use to consume your service/resource

Example:

```
OPTIONS /resource/foo

Access-Control-Request-Method: DELETE

Access-Control-Request-Headers: origin, x-requested-with

Origin: https://foo.bar.org
```