Web Programming

Cookie, Proxy, and Cache

Mahsa Saeidi

Cookies

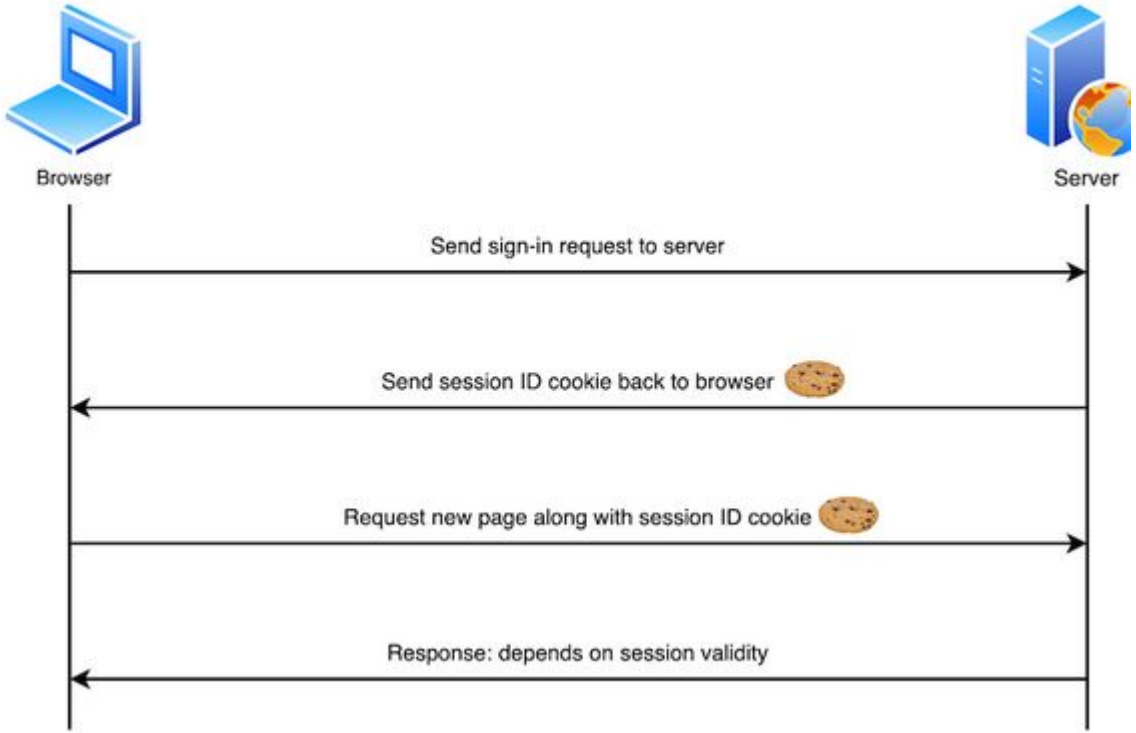
Stateless Problem

- HTTP is a stateless protocol
  - Server does not remember it's client
- How to personalize pages (personal portal)?
  - Use http header: X-Forwarded-For
    - Is not usually sent by browsers
  - Find client IP address from TCP connection
    - The problem is NAT
  - Clients move but IP address does not

Solution of Stateless Problem: Cookie [RFC 6265]

- Cookie: information (e.g. unique identifiers) sent by server to user (browser) which are returned back to server
- How it works
  - Server asks client to remember the information
    - Set-Cookie header in response message
    - Set-Cookie: <cookie-name>= <cookie-value>
  - Client gives back the information to server in every request
    - Cookie header in request messages
    - Cookie: <cookie-name>= <cookie-value>; <cookie-name>= <cookie-value>
  - Server customizes responses according to the cookie

Cookies By an Example



Cookies By an Example

1. The user sends **sign-in credentials** to the server, for example via a form submission.
2. If the credentials are correct, the server updates the UI to indicate that the user is signed in, and responds with a **cookie** containing a **session ID** that records their **sign-in status** on the browser.
3. At a later time, the user moves to a different page on the same site. The browser sends the **cookie** containing the **session ID** along with the corresponding request to indicate that it still thinks the user is signed in.
4. The server checks the **session ID** and, if it is still valid, sends the user a personalized version of the new page. If it is not valid, the session ID is deleted and the user is shown a generic version of the page (or perhaps shown an "access denied" message and asked to sign in again)

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

The Purpose of Cookies

- **Session management:** User sign-in status, shopping cart contents, game scores, or any other user session-related details that the server needs to remember.
- **Personalization:** User preferences such as display language and UI theme.
- **Tracking:** Recording and analyzing user behavior.

Types of Cookies

- **Session cookies:**
  - To identify a session
  - session cookie only lasts for the duration of users using the website.
  - A session cookie expires if the user does not access the website for a period of time chosen by the server (idle timeout).
  - **Example**: shopping cart on eCommerce websites

- **Persistent cookies:**
  - To identify a client (browser)
  - Persistent Cookies are stored on a user's device to retain user’s information, preferences, settings, or login details .
  - If a persistent cookie has its maximum age 1 year, then the original value stored in the cookie will be sent back to the server each time the user accesses the server within that year.
  - **Example:** Gmail Login — remember me

Cookies

- A browsers should be able to accept:
  - At least 300 cookies
  - At least 4096 bytes per cookie
  - At least 20 cookies per unique host or domain name
- Cannot be used to store large data
- Cookies are text files
  - No virus spread
- There is not any request from server to read cookies
  - By default cookies are sent by browser
  - Browser checks URL and finds appropriate cookies

Cookies

- Client can control cookies
  - Disable cookies: no cookie is saved & used
  - View & Delete cookies
- Server can control cookies by its attributes
  - Expiration time
  - Domain
  - Path
  - Security
  - ...

Cookies Attributes

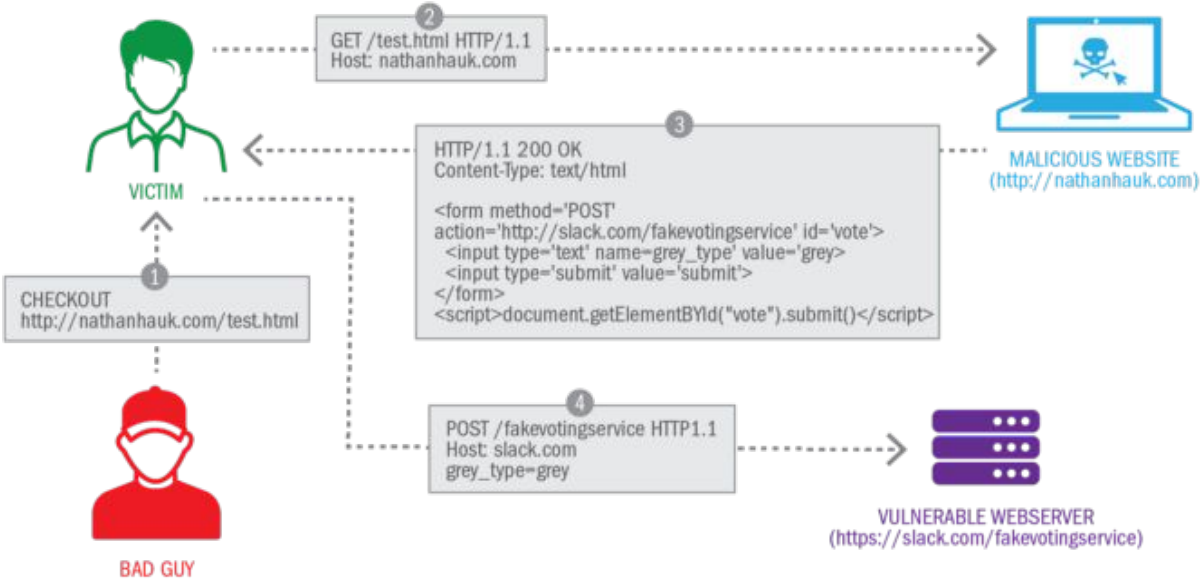
- **Expire & Max-Age:** The life time of the cookie
  - **Expire**: An absolute time to delete cookie
  - **Max-Age**: The maximum life time (sec) of cookie
  - If exists, shows a permanent cookie
  - If does not exist, shows a session cookie and will be expired when session ends (more)
  - Send a past time (or negative) to delete a cookie
- **Secure**: Cookie is sent only if channel is secure
  - Specially useful for login sessions cookies
  - A cookie with the Secure attribute is sent to the server only with an encrypted request over the **HTTPS protocol**, never with unsecured HTTP (except on localhost), and therefore can't easily be accessed by a man-in-the-middle attacker.

Cookies Attributes

- **HttpOnly:** Cookie is sent only if HTTP is used
  - Cookies can not be read by any malicious script running on the website mitigating the session hijacking
- **SameSite:** allows you to declare if your cookie should be restricted to a first-party or same-site context
  - **Lax**: Cookies will only be sent when users actively click a link to a third-party website. All the sites belonging to the same domain can set and access cookies.
  - **Strict**: Restricts cross-site sharing altogether.
  - Cookies with this setting can be accessed only when visiting the domain from which it was initially set. In other words, Strict completely blocks a cookie being sent to a.com when a page from b.com makes the request.
  - **None**: Cookies will be sent in all contexts, i.e in responses to both first-party and cross-origin requests. If SameSite=None is set, the cookie Secure attribute must also be set (or the cookie will be blocked).

CSRF

Cross-site request forgery (CSRF) attacks rely on the fact that cookies are attached to any request to a given origin, no matter who initiates the request.



In-Depth Defence

- SameSite (Cookie Attribute)
- Warning!

  Neither Strict nor Lax are a complete solution for your site's security. Cookies are sent as part of the user's request and you should treat them the same as any other user input. That means sanitizing and validating the input. Never use a cookie to store data you consider a server-side secret.

Cookies Attributes: Domain & Path

- **Domain & Path** determine the scope of the cookie
  - For which path and domain, the cookie is saved & returned back by browser
- If Domain is omitted, defaults to the host of the current document URL.
  - Browser returns back the cookie for the domain and not sub-domains
- If Path is omitted, defaults to the path of the current document URL.
  - Browser returns back the cookie for the path and also for all sub-paths
- If present then browser checks validity
  - If they are valid then Browser returns back the cookie for that domain & that path and also for all sub-domains and sub-path

Cookies Attributes: Domain & Path

- Validity check by major browsers
  - Domain names must start with dot
    - Some browsers accept names without dot as domain
    - Contrary to earlier specifications, leading dots in domain names (.example.com) are ignored.
  - Don’t accept for other domains than the base domain
  - Don’t accept cookies for sub-domains
  - Accept cookies for higher domains
    - Except the top level domains, e.g., .com, .ac.ir
  - Accept cookies for other (sub or higher) paths
    - A path that must exist in the requested URL, or the browser won't send the Cookie header.

Session Hijacking

- Sending cookies over unencrypted HTTP is a very bad idea
  - Attackers may steal the user’s session cookie, locate the session ID within the cookie, and use that information to take over the session
- Attacker sends victim's cookie as if it was their own
- Server will be fooled

Session Hijacking (Cont.)

- Web attacks that would cause accessing to cookies by attackers
  - **Cross-Site Scripting (XSS):** attackers inject malicious scripts into web pages viewed by other users. (e.g., if a website allows user input to be displayed unfiltered (like in comments or posts), an attacker can insert a script that steals cookies from anyone viewing the content.
  - **Buffer overflow attack (server side)**: allows attackers to execute a code and potentially access to cookie data stored in memory.

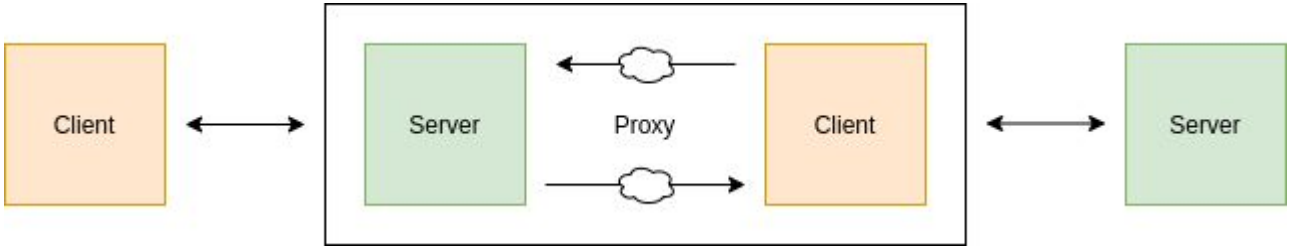
Defence

- HTTPOnly
- HTTPS
- ...

Proxy

Proxy

- Proxies sit between client and server
- Act as server for client
- Act as client for server

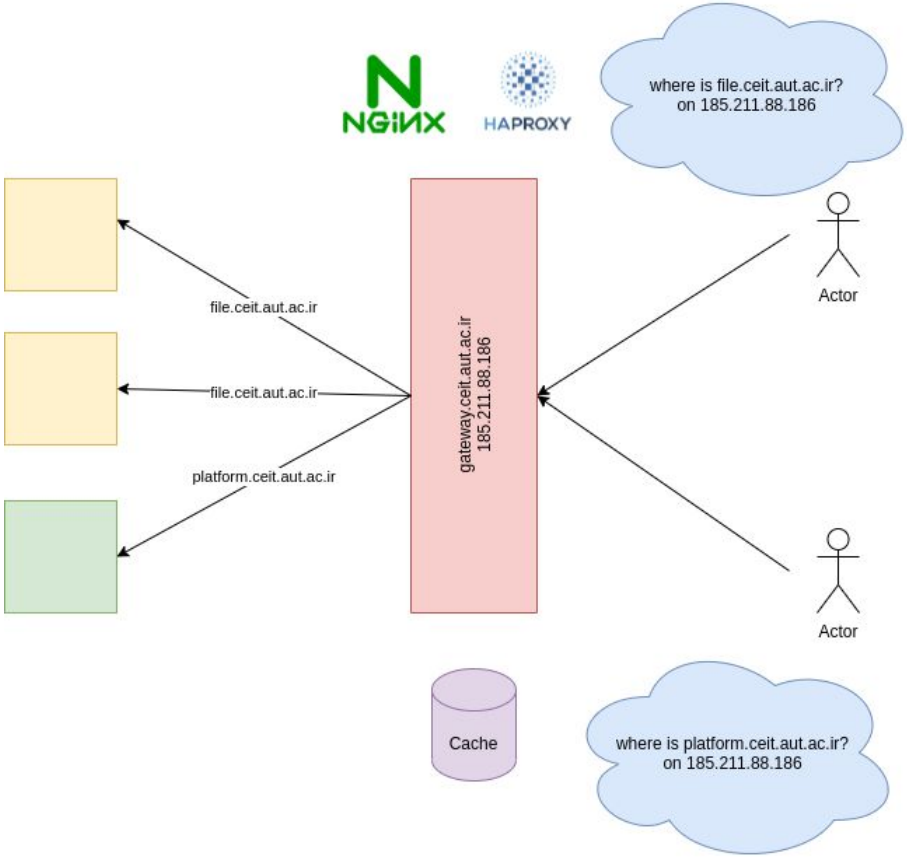


Forward Proxies

- A forward proxy, or gateway, or just "proxy" provides proxy services to a client or a group of clients.

Reverse Proxies

- As the name implies, a reverse proxy does the opposite of what a forward proxy does:
- A forward proxy acts in behalf of clients (or requesting hosts), a reverse proxy acts in behalf of servers.
- Forward proxies can hide the identities of clients whereas reverse proxies can hide the identities of servers.
- Reverse proxies have several use cases, a few are:
    a. Load balancing: distribute the load to several web servers,
    b. Cache static content: offload the web servers by caching static content like pictures,
    c. Compression: compress and optimize content to speed up load time.


NGINX
HAPROXY
where is file.ceit.aut.ac.ir?
on 185.211.88.186
file.ceit.aut.ac.ir
file.ceit.aut.ac.ir
platform.ceit.aut.ac.ir
gateway.ceit.aut.ac.ir
185.211.88.186
Actor
Actor
Cache
where is platform.ceit.aut.ac.ir?
on 185.211.88.186

HTTP Proxy Applications

- Authentication
  - Client side: Authenticate clients before they access web
  - Server side: Authenticate clients before access the server
- Accounting: Log client activities
  - Security: Analyze request before sending it to server
- Integrated in modern firewalls
  - Filtering: Limit access to specified contents
  - Anonymizer: Anonymous web browsing
  - Caching (more details in the following slide)

Cache

Caching

- Caching: save a copy of a resource and use it instead of requesting server
- Browser has its own local caches
- Cache server is special proxy for caching
- Benefits
  - Reduce redundant data transfer
  - Reduce network bottleneck
  - Reduce load on server
  - Reduce delay

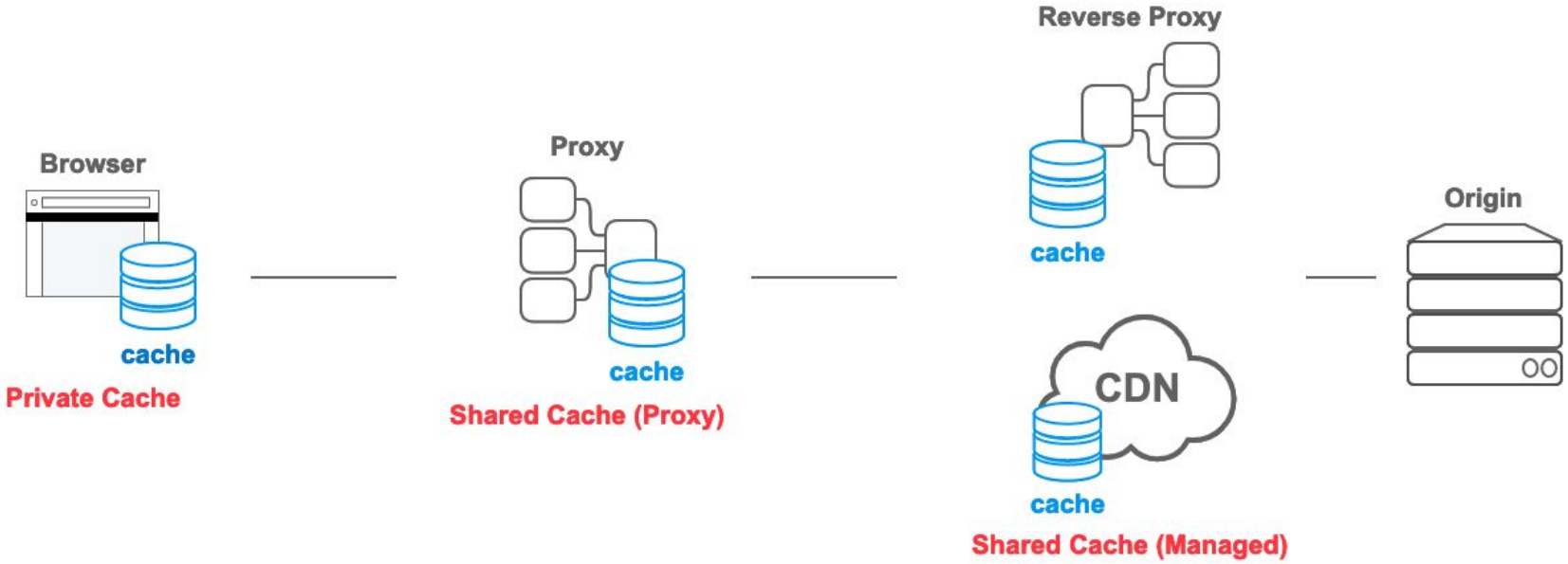
Cache Types

- Private caches
  - tied to a specific client — typically a browser cache.
  - can store a personalized response for that user.
  - If a response contains personalized content and you want to store the response only in the private cache, you must specify a private directive.
    - Cache-Control: private

Cache Types (Cont.)

- Shared Cache
  - The shared cache is located between the client and the server and
  - can store responses that can be shared among users.
  - shared caches can be further sub-classified into
    - proxy caches
      - some proxies implement caching to reduce traffic out of the network.
      - Not very common when HTTPS has become more common
    - managed caches.
      - deployed by service developers to offload the origin server and to deliver content efficiently.
        - E.g., reverse proxies, CDNs, and service workers in combination with the Cache API.

Cache Types (Cont.)



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

Fresh and Stale

- State of Stored HTTP responses:
  - Fresh: The fresh state usually indicates that the response is still valid and can be reused,
- Stale: the stale state means that the cached response has already expired.
- Age: determines when a response is fresh or stale.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Cache-Control: max-age=604800
```

Validation

- A HTTP mechanism to transform a stale response into a fresh one by asking the origin server.
  - Validation
  - Revalidation
- Using a conditional request
  - **If-Modified-Since header**
    - Asking “if there have been any changes made since the specified time”
  - **If-None-Match request header**
    - Puts the value of the **ETag** response header for the cached response into the **If-None-Match** request header, to ask the server if the resource has been modified

If-Modified-Since header

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
```

```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-Modified-Since: Tue, 22 Feb 2022 22:00:00 GMT
```

```
HTTP/1.1 304 Not Modified
Content-Type: text/html
Date: Tue, 22 Feb 2022 23:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
```

ETag/If-None-Match

- An arbitrary value generated by the server.
  - a hash of the body contents or
  - a version number.
- The server will return **304 Not Modified** if the value of the ETag header is the same as the **If-None-Match** value in the request.
- But if the server determines the requested resource should now have a different **ETag** value, the server will instead respond with **a 200 OK** and the latest version of the resource.

ETag/If-None-Match

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
ETag: "33a64df5"
Cache-Control: max-age=3600
```

```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-None-Match: "33a64df5"
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 23:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
ETag:      "dfggs34h"
```

Don’t Cache

- The **no-cache** directive does not prevent the storing of responses but instead prevents the reuse of responses without revalidation.
- Use **no-store** to prevent the response from being stored in any cache.

```
Cache-Control: no-store
```

Caching Algorithm in Summary

- If the object is not cached, it is got from server, saved in cache, and sent to client
- Else, if object is in cache
  - Cache server must return only fresh objects
  - Freshness check
- Objects life-time specified by server
- The **Cache-Control** HTTP/1.1 general-header field is used to specify directives for caching mechanisms in both requests and responses.

Caching Algorithm in Summary (Cont.)

- Expiration
  - the maximum amount of time a resource will be considered fresh.
    - Cache-Control: max-age=<seconds>
- No caching
  - The cache should not store anything about the client request or server response.
    - Cache-Control: no-store
- Cache but revalidate
  - A cache will send the request to the origin server for validation before releasing a cached copy.
    - Cache-Control: no-cache

Caching Algorithm in Summary (Cont.)

- If requested object is not expired
  - Cache server gives it to client
- If requested object is expired
  - Its freshness must be checked
- Freshness is checked by conditional request
  - If-Modified-Since: current last-modified time
  - If-None-Match: the server will send back the requested resource, with a 200 status, only if it doesn't have an ETag matching the given ones.
    - The ETag HTTP response header is an identifier for a specific version of a resource.

Caching Algorithm in Summary (Cont.)

- Server responses
  - 304 Not modified response + new expire time
    - Cached copy is valid until the specified time
  - 200 OK
    - Server provides a new version of the object
    - Cache server updates cached copy

Authentication

Authentication vs Authorization

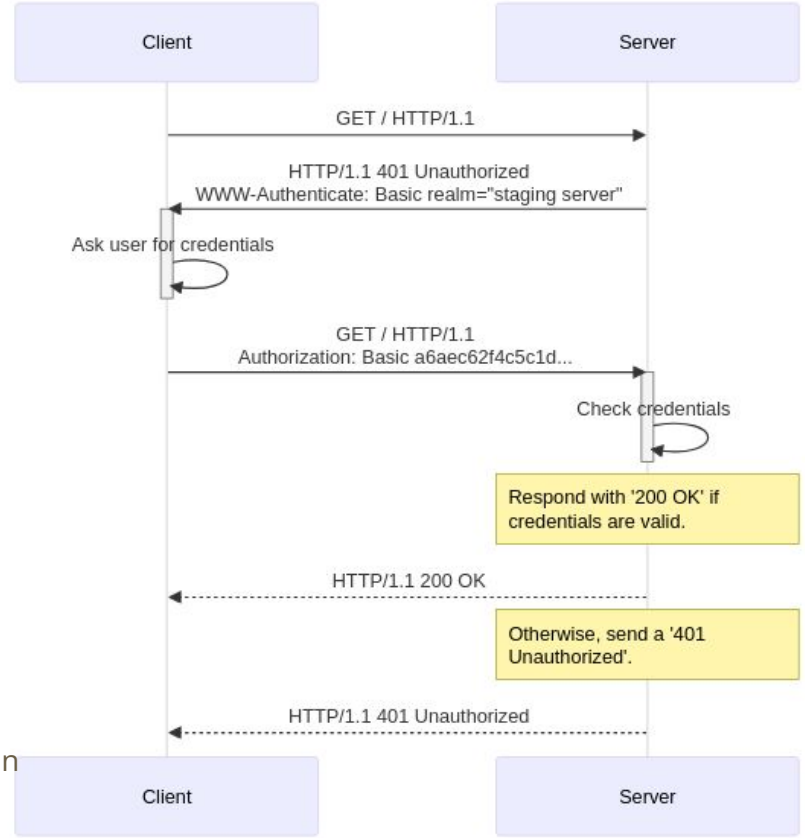
- Authentication is the verification of the credentials of the connection attempt.
- This process consists of sending the credentials from the remote access client to the remote access server in an either plaintext or encrypted form by using an authentication protocol.
- Authorization occurs after successful authentication determining the user's privileges and grants access accordingly

HTTP Authentication

- All resources are not public in web; e.g.,
  - Financial documents, Customer information, ...
- HTTP has two (similar) authentications
  - **Basic: Base64 encoded user:pass** [The username itself cannot contain a colon]
  - **Digest: Plain username + Digest of pass**
- Steps are the same
  - Client-side app (browser) request resource from server
  - Server refuses with 401 Unauthorized
  - Client-side app ask Username & Password from user
  - Client send Username & Password to server
  - Server authenticates and allows.
- Authentication information are sent by every request until end of current session

HTTP Authentication Message Flow

1. The server responds to a client with a **401 (Unauthorized) response status** and provides information on how to authorize with a WWW-Authenticate response header containing at least one challenge.
2. A client that wants to authenticate itself with the server can then do so by including an Authorization request header with the credentials.
3. Usually a client will present **a password prompt** to the user and will then issue the request including the correct Authorization header.



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

Base64

- Base64 is a group of binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation.

Basic Authentication

- A server responses with a **WWW-Authenticate** response header supporting **basic authentication**.
- A user-agent receiving this header would first **prompt** the user for their username and password, and then re-request the resource including the (encoded) **credentials** in the **Authorization header**.
- For **"Basic" authentication** the credentials are constructed by first combining the username and the password with a colon (aladdin:opensesame), and then by **encoding** the resulting string in **base64** (YWxhZGRpbjpvcGVuc2VzYW1l)

Digest authentication with SHA-256 and MD5

- The client attempts to access a document at URI **http://www.example.org/dir/index.html** that is protected via **digest authentication.**
- The username for this document is "Mufasa" and the password is "Circle of Life"

Digest authentication with SHA-256 and MD5

- The server responds with an **HTTP 401 message** that includes a **challenge** for each **digest algorithm** it supports, (**SHA256** and **MD5**)

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
    realm="http-auth@example.org",
    qop="auth, auth-int",
    algorithm=SHA-256,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
WWW-Authenticate: Digest
    realm="http-auth@example.org",
    qop="auth, auth-int",
    algorithm=MD5,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

Digest authentication with SHA-256 and MD5

- The client prompts the user for their username and password, and then responds with a new request that encodes the credentials in the **Authorization header** field.

  HA1=MD5(username,realm,password)
  HA2 =MD5(method,URI)

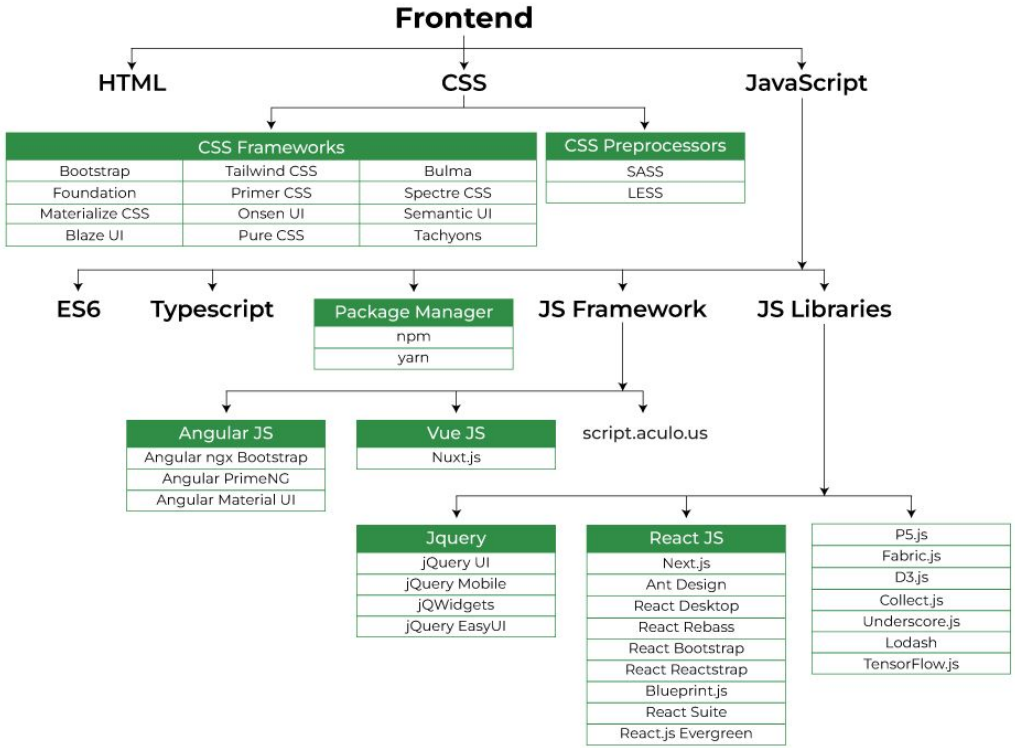
  returns
  MD5(HA1,nonce,nc,cnounce,HA2)

```
Authorization: Digest username="Mufasa",
    realm="http-auth@example.org",
    uri="/dir/index.html",
    algorithm=MD5,
    nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
    nc=00000001,
    cnonce="f2/wE4q74E6zIJEtWaHKaf5wv/H5QzzpXusqGemxURZJ",
    qop=auth,
    response="8ca523f5e9506fed4657c9700eebdbec",
    opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

References

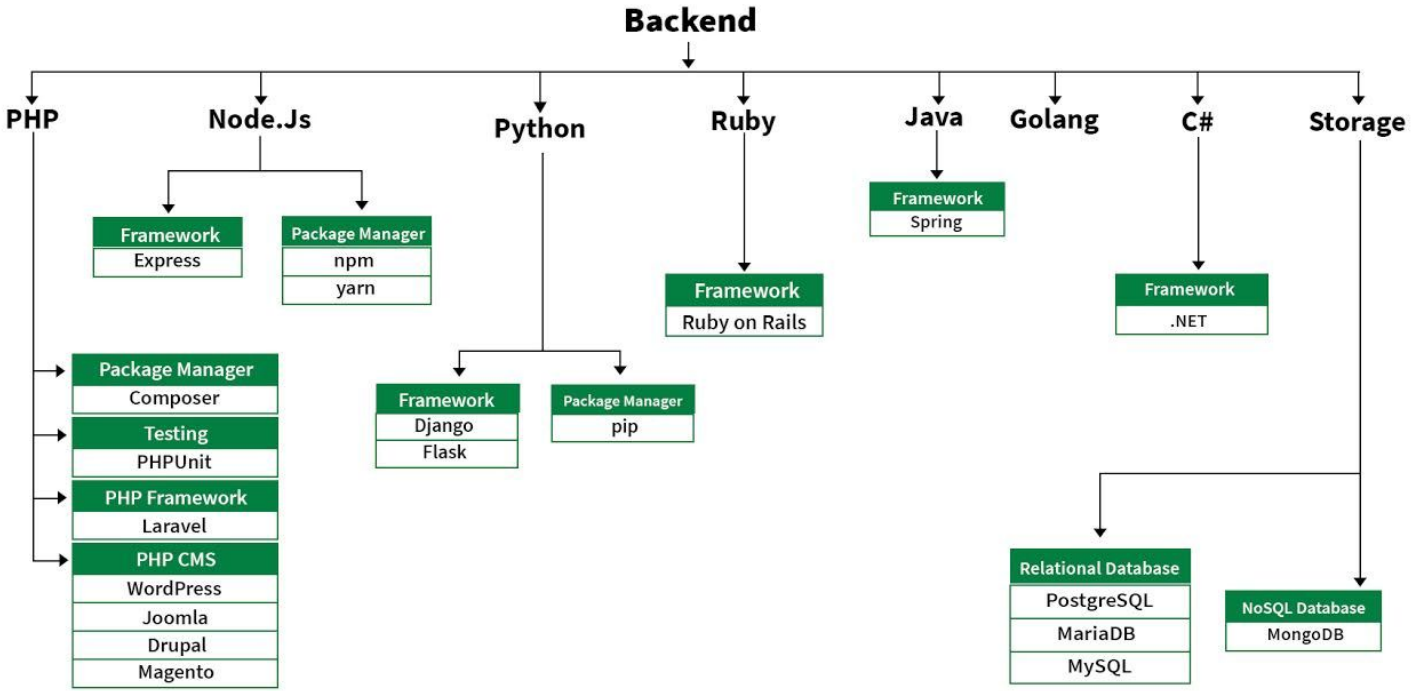
- The majority of these slides have been adapted from materials provided by other instructors:
  - lecture notes developed by Mr. Parham Alvani
  - Internet Eng. Course's Slides developed by Dr.Bahador Bakhshi
- Other Resources
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/

Web Development Roadmap



Source: https://www.geeksforgeeks.org/web-development/

Web Development Roadmap



Source: https://www.geeksforgeeks.org/web-development/