Série modulaire sur les

Systèmes distribués

Fascicule 1:

Le protocole http

Bernard Beaulieu

Université de Sherbrooke

Février 2022

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document fascicule01.pdf
Version 1.0, Février 2022

Département de génie électrique et de génie informatique, Université de Sherbrooke.
Réalisé à l'aide de LaTeX et TeXstudio.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Chapitre 1

Protocole http

HTTP est un protocole qui permet de récupérer des ressources telles que des documents HTML, du texte non formattée, du transfert d'information en json ou xml. Il est à la base de tout échange de données sur le Web. C'est un protocole de type client-serveur, ce qui signifie que les requêtes sont initiées par le destinataire (navigateur, commande curl). Un document complet est construit à partir de différents sous-documents provernant du serveur Web et déchiffrer en partie par le message. Les clients et serveurs communiquent par l'échange de messages individuels (en opposition à un flux de données). On appelle requête les messages envoyés par le client et réponses les messages renvoyés par le serveur. Conçu au début des années 1990, HTTP est un protocole extensible qui a évolué au cours du temps. C'est un protocole de la couche application dont les données transitent via TCP ou à travers une connexion TCP chiffrée avec TLS. En théorie, tout protocole de transport fiable pourrait être utilisé. Dans le transfert de messages entre un client et un serveur, il y a plusieurs composantes qui entre en jeu: il y a les switches, routeurs, modems et bien plus. Grâce à la construction en couche du Web, ces intermédiaires sont cachés dans les couches réseau et transport. HTTP est bâti sur la couche applicative. Bien qu'elles puissent s'avérer importantes lorsqu'il s'agit de diagnostiquer des problèmes réseau, les couches inférieures ne sont pas pertinentes ici pour décrire HTTP.

1.1 Principaux aspects du protocole HTTP

1.1.1 HTTP est simple

Même s'il est devenu plus complexe avec l'arrivée de la version HTTP/2 et l'encapsulation des messages HTTP dans des trames, HTTP est généralement conçu pour être simple et lisible par un humain. Les messages HTTP peuvent être lus, ce qui facilite les tests des développeurs et réduit la complexité pour les débutants.

1.1.2 HTTP est extensible

À partir de HTTP/1.0, les en-têtes HTTP permettent d'étendre facilement le protocole et faire des expérimentations avec celui-ci. De nouvelles fonctionnalités peuvent même être introduites par un simple accord entre le client et le serveur à partir de la sémantique dans de nouveaux en-têtes.

1.1.3 HTTP est sans état, mais pas sans session

HTTP est sans état : il n'y a pas de lien entre deux requêtes qui sont effectuées successivement sur la même connexion. Cela devient très rapidement problématique lorsque les utilisateurs veulent interagir avec une page de façon cohérente, par exemple avec un panier d'achat sur un site de commerce en ligne. Cependant, bien que le coeur d'HTTP soit sans état, on peut utiliser les cookies et les sessions pour palier à cet aspect Nous reviendrons plus loin avec ces deux notions.

1.1.4 HTTP et les connexions

Une connexion est contrôlée au niveau de la couche transport et est donc fondamentalement hors de portée du protocole HTTP. Le protocole doit donc être fiable pour empêcher la perte d'information. Il doit donc gérer au minimum la remontée des erreurs. Parmi les deux protocoles de transport les plus courants sur Internet, TCP est fiable et UDP ne l'est pas. Ainsi, HTTP s'appuie donc sur le standard TCP. HTTP/1.0 ouvre une connexion TCP pour chaque échange requête/réponse, ce qui introduit un problème majeur de lenteur. Puisque l'ouverture d'une connexion nécessite plusieurs allers-retours, la réponse aux clients se fait souvent attendre. Afin de réduire ce défaut, HTTP/1.1 introduit le pipelining (qui s'est avéré difficile à mettre en oeuvre) et les connexions persistantes : la connexion TCP sous-jacente peut être partiellement contrôlée en utilisant l'en-tête Connection. HTTP/2 va plus loin en multiplexant des mes-

sages sur une seule connexion, ce qui aide à maintenir une relative rapidité de la réponse aux clients.

1.2 Efficacité du protocole HTTP

Au fil du temps, la nature extensible de HTTP lui a permis d'y ajouter de nouvelles fonctionnalités. Les méthodes de cache ou d'authentification sont des fonctions qui furent gérées dès le début de HTTP tandis que la possibilité de lever la contrainte d'unicité de l'origine (cross origin) ne fut introduite qu'à partir des années 2010.

Voici une liste de fonctionnalités courantes, qui peuvent être contrôlées grâce à HTTP.

- Cache web : Les performances des sites et applications web peuvent être significativement améliorées en réutilisant les ressources déjà collectées précédemment. C'est ce qu'on appelle la cache web. Les caches réduisent la latence et le trafic du réseau, et ainsi diminuent le temps nécessaire à l'affichage de la représentation d'une ressource. En utilisant la mise en cache HTTP, les sites web deviennent plus réactifs La façon dont les documents sont mis en cache peut être contrôlée par le protocole HTTP. Ça donne la possibilité au serveur d'indiquer aux clients ce qu'ils doivent et peuvent mettre en cache et pour combien de temps.

- Lever la contrainte d'origine unique (Cross origin) : Pour éviter l'espionnage et d'autres invasions dans la vie privée, le protocole peut imposer une séparation stricte entre les sites web. Ainsi seules les pages provenant d'un même origine (même adresse réseau ou sous-réseau) peuvent accéder à toutes les informations d'une autre page web. Bien que cette contrainte soit un fardeau pour le serveur, les en-têtes HTTP peuvent assouplir cette séparation stricte du côté serveur, en permettant à un document de devenir un ramassi d'informations en provenance de différents domaines (il existe même des raisons de sécurité de procéder ainsi).

- Authentification : Certaines pages peuvent être protégées de sorte que seuls certains utilisateurs puissent y accéder. Une authentification simple peut être fournie par HTTP, soit en utilisant l'en-tête WWW-Authenticate et des en-têtes similaires, soit en définissant une session spécifique grâce à des cookies HTTP.

- Cookie : Un cookie HTTP ou témoin de connexion est un petit texte envoyé par un serveur à un client. Ce dernier le renverra la prochaine fois qu'il se connectera aux serveurs partageant le même nom de domaine. Un cookie a une date et un heure de péremption. C'est-à-dire que lorsqu'il est échue, il n'est plus utilisable.

- Sessions : L'utilisation de cookies HTTP permet de lier les requêtes du client à l'état du serveur. Cela crée des sessions, malgré le fait que le protocole HTTP soit, au sens strict, un protocole sans état. Ceci est utile dans le cas du commerce électronique en ligne et aussi pour tous les site permettant une configuration de l'utilisateur. Également les cookies permettent, lorsqu'on est sur un site ou l'authetification est obligatoire, d'éviter la réauthentifications à chaque changement de page provenant du même nom de domaine.

1.3 Flux HTTP

Lorsqu'un client veut communiquer avec un serveur, il réalise les étapes suivantes :

1. Il ouvre une connexion TCP : la connexion TCP va être utilisée pour envoyer une ou plusieurs requêtes et pour recevoir une réponse. Le client peut ouvrir une nouvelle connexion, réutiliser une connexion existante ou ouvrir plusieurs connexions TCP vers le serveur.
2. Il envoie un message (requête) HTTP : les messages HTTP (avant HTTP/2) sont lisibles par les humains. Avec HTTP/2, ces simples messages sont en-capsulés dans des trames, rendant la lecture directe impossible, mais le principe reste le même.

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Algorithme 1.1: Envoi au serveur

3. Il lit la réponse envoyée par le serveur :

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (suivi des 29769 octets de la page web demandée)
```

Algorithme 1.2: Réponse du serveur

4. Il ferme ou réutilise la connexion pour les requêtes suivantes.

1.4 Les messages HTTP

Les messages HTTP/1.1 et ceux des versions précédentes d'HTTP sont lisibles par des humains. Avec HTTP/2, ces messages sont intégrés dans une nouvelle structure binaire, une trame, ce qui permet des optimisations telles que la compression des en-têtes et le multiplexage. Même si seule une partie du message HTTP d'origine est envoyée dans cette version d'HTTP, la sémantique de chaque message est inchangée et le client reconstitue (virtuellement) la requête HTTP/1.1 d'origine. Il est donc utile de comprendre les messages HTTP/2 au format HTTP/1.1.

Il existe deux types de messages HTTP, les requêtes et les réponses, chacun ayant son propre format.

1.4.1 Requêtes

Voici un exemple de requête HTTP :

Une requête comprend les éléments suivants :

- Une méthode HTTP : généralement un verbe tel que GET, POST ou un nom comme OPTIONS ou HEAD qui définit l'opération que le client souhaite effectuer. Par exemple, un client souhaite accéder à une ressource (en utilisant GET) ou téléverser le résultat d'un formulaire HTML (en utilisant POST).

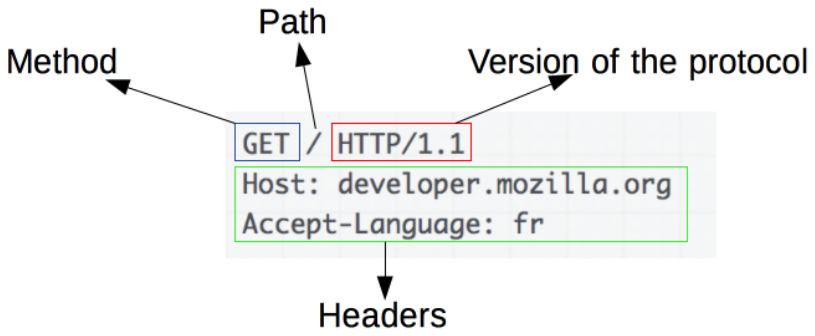

Figure 1.1: Exemple de requête http

- Le chemin de la ressource à extraire : l'URL de la ressource à laquelle on a retiré les éléments déductibles du contexte, par exemple le protocole (http://)le domaine (ici .mozilla.org), ou le port TCP (ici 80).
- La version du protocole HTTP.
- Les en-têtes optionnels qui transmettent des informations supplémentaires pour les serveurs.
- Ou un corps, pour certaines méthodes comme POST, semblable à ceux dans les réponses, qui contiennent la ressource envoyée.

1.4.2 Réponses

Voici un exemple de réponse :

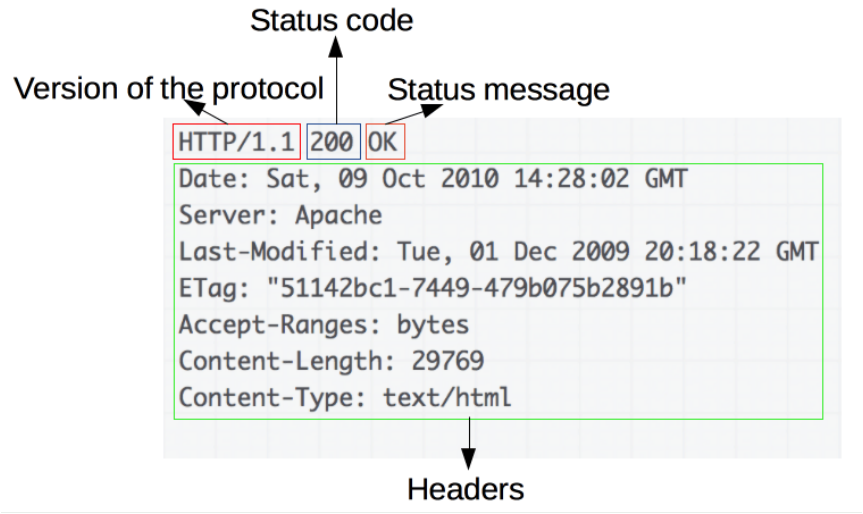

Figure 1.2: Exemple de réponse http

Une réponse comprend les éléments suivants:

- La version du protocole HTTP qu'elle suit
- Un code de statut, qui indique si la requête a réussi ou non.
- Un message de statut qui est une description rapide, informelle, du code de statut
- Les en-têtes HTTP, comme pour les requêtes.
- Éventuellement un corps contenant la ressource récupérée.

1.5 Les codes de statut de réponse HTTP

Les codes de statut de réponse HTTP indiquent si une requête HTTP a été exécutée avec succès ou non. Les numéros des codes varient de 100 à 599. On peut les regroupés en cinq classes :

- Les réponses informatives (100 - 199),
- Les réponses de succès (200 - 299),
- Les messages de redirection (300 - 399),
- Les erreurs du client (400 - 499),
- Les erreurs du serveur (500 - 599).

1.5.1 Réponses informatives

100 Continue

Cette réponse intermédiaire indique que tout est OK pour le moment et que le client peut continuer sa requête ou l'ignorer si celle-ci est déjà finie.

101 Switching Protocols

Ce code est envoyé en réponse à un en-tête de requête Upgrade (en-US) de la part du client et indique le protocole sur lequel passe le serveur.

102 Processing (WebDAV)

Ce code indique que le serveur a reçu et est en train de traiter la requête mais qu'une réponse n'est pas encore disponible.

103 Early Hints

Ce code de statut est conçu pour être utilisé avec l'en-tête Link (en-US), ce qui permet à l'agent utilisateur de commencer le préchargement des ressources tandis que le serveur prépare une réponse.

1.5.2 Réponses de succès

200 OK

La requête a réussi. La signification du succès peut varier selon la méthode HTTP :

- GET : La ressource a été récupérée et est retransmise dans le corps du message.
- HEAD : Les en-têtes d'entité sont présents dans la réponse et il n'y a pas de corps.
- PUT ou POST : La ressource décrivant le résultat de l'action est transmise dans le corps du message.
- TRACE : Le corps du message contient le message de requête tel que reçu par le serveur.

201 Created

La requête a réussi et une nouvelle ressource a été créée en guise de résultat. Il s'agit typiquement de la réponse envoyée après une requête PUT ou POST.

202 Accepted

La requête a été reçue mais n'a pas encore été traitée. C'est une réponse évasive, ce qui signifie qu'il n'y a aucun moyen en HTTP d'envoyer une réponse asynchrone ultérieure indiquant le résultat issu du traitement de la requête. Elle est destinée aux cas où un autre processus ou serveur gère la requête, et peut être utile pour faire du traitement par lots.

203 Non-authorative Information

Ce code de réponse signifie que l'ensemble de méta-informations renvoyé n'est pas exactement l'ensemble disponible sur le serveur d'origine, mais plutôt un ensemble collecté à partir d'une copie locale ou tierce. Ce code est utilisé la plupart du temps par les serveurs miroirs ou de sauvegarde d'une autre ressource. À l'exception de cette condition, une réponse 200 OK est préférable.

204 No Content

Il n'y a pas de contenu à envoyer pour cette requête, mais les en-têtes peuvent être utiles. L'agent utilisateur peut mettre à jour ses en-têtes en cache pour cette ressource en les remplaçant par les nouveaux.

205 Reset Content

Indique à l'agent utilisateur de réinitialiser le document qui a envoyé cette requête.

206 Partial Content

Ce code de réponse est utilisé en réaction à l'en-tête Range (en-US) envoyé par le client pour séparer le téléchargement en plusieurs flux.

207 Multi-Status

Une réponse multi-statut donne des informations sur des ressources multiples dans les situations où les codes de statut multiples sont appropriés.

208 Already Reported

Utilisé au sein d'un élément de réponse DAV <dav:propstat> pour éviter d'énumérer à maintes reprises les membres internes de bindings multiples vers la même collection.

226 IM Used (HTTP Delta encoding)

Le serveur a exécuté une requête GET pour la ressource, et la réponse est une représentation du résultat d'une ou plusieurs manipulations d'instance appliquées à l'instance courante.

1.5.3 Message de redirection

300 Multiple Choices

La requête a plusieurs réponses possibles. Le client doit choisir l'une d'entre elles. Il n'y a pas de manière standard pour choisir l'une de ces réponses mais des liens HTML vers les choix sont recommandés afin de permettre à l'utilisateur de choisir.

301 Moved Permanently

Ce code de réponse signifie que l'URL de la ressource demandée a été modifiée. Une nouvelle URL est donnée dans la réponse.

302 Found

Ce code de réponse indique que l'URI de la ressource demandée a été modifiée temporairement. De nouveaux changements dans l'URI pourront être effectués ultérieurement. Par conséquent, cette même URI devrait être utilisée par le client pour les requêtes futures.

303 See Other

Le serveur a envoyé cette réponse pour diriger le client vers la ressource demandée via un autre URI en utilisant une requête GET.

304 Not Modified

Ce code est utilisé pour des raisons de cache. Il indique au client que la réponse n'a pas été modifiée. De fait, le client peut continuer à utiliser la même version de la réponse, mise en cache.

305 Use Proxy

A été défini dans une version antérieure de la spécification HTTP pour indiquer qu'une réponse sollicitée doit transiter par un proxy. Ce code est aujourd'hui périmé pour des raisons de sécurité relatives à la configuration d'un proxy.

306 unused Ce code de réponse n'est plus en service, son usage est actuellement réservé. Il était utilisé dans une version précédente de la spécification HTTP/1.1.

307 Temporary Redirect

Le serveur a envoyé cette réponse pour rediriger le client afin d'obtenir la ressource demandée via un autre URI, en utilisant la même méthode que précédemment. Ce code a la même sémantique que le code 302 Found, à l'exception près que l'agent utilisateur ne doit pas changer la méthode HTTP utilisée : si POST était utilisé dans la première requête, alors POST doit être utilisé dans la seconde.

308 Permanent Redirect

Cela signifie que la ressource a été déplacée de manière permanente vers une autre URI, spécifiée dans l'en-tête de réponse HTTP Location:. Ce code a la même sémantique que le code 301 Moved Permanently, à l'exception près que l'agent utilisateur ne doit pas changer la méthode HTTP utilisée : si POST était utilisé dans la première requête, alors POST doit être utilisé dans la seconde.

1.5.4 Réponses d'erreur côté client

- 400 Bad Request

  Cette réponse indique que le serveur n'a pas pu comprendre la requête à cause d'une syntaxe invalide.

- 401 Unauthorized

  Bien que le standard HTTP indique ń non-autorisé ż, la sémantique de cette réponse correspond à ń non-authentifié ż : le client doit s'authentifier afin d'obtenir la réponse demandée.

- 402 Payment Required

  Ce code de réponse est réservé à une utilisation future. Le but initial justifiant la création de ce code était l'utilisation de systèmes de paiement numérique. Cependant, il n'est pas utilisé actuellement et aucune convention standard n'existe à ce sujet.

- 403 Forbidden

  Le client n'a pas les droits d'accès au contenu, donc le serveur refuse de donner la véritable réponse.

- 404 Not Found

  Le serveur n'a pas trouvé la ressource demandée. Ce code de réponse est principalement connu pour son apparition fréquente sur le web.

- 405 Method Not Allowed

  La méthode de la requête est connue du serveur mais n'est pas prise en charge pour la ressource cible. Par exemple, une API peut ne pas autoriser l'utilisation du verbe DELETE pour supprimer une ressource.

- 406 Not Acceptable

  Cette réponse est envoyée quand le serveur web, après une négociation de contenu géré par le serveur, ne trouve rien qui satisfasse les critères donnés par l'agent utilisateur.

- 407 Proxy Authentication Required Similaire au code 401, sauf que l'authentification doit être effectuée au travers d'un proxy.

- 408 Request Timeout Cette réponse est envoyée via une connexion en attente par certains serveurs, même sans qu'il y ait de requête préalable de la part du client. Cela signifie que le serveur aimerait fermer cette connexion inutilisée. Cette réponse est bien plus utilisée depuis que certains navigateurs, comme Chrome, Firefox 27+ ou IE9, utilisent des mécanismes de préconnexion HTTP pour accélérer la navigation. Notez aussi que certains serveurs ferment simplement la connexion sans même envoyer ce message.

- 409 Conflict Cette réponse est envoyée quand une requête entre en conflit avec l'état actuel du serveur.

- 410 Gone Cette réponse est envoyée lorsque le contenu demandé a été supprimé de façon permanente du serveur, sans nouvelle adresse. Les clients doivent vider les caches et liens associés à cette ressource. La spécification HTTP a conçu ce code de statut pour qu'il soit utilisé pour des ń services promotionnels limités dans le temps ż. Les API ne devraient pas se sentir obligées d'indiquer que des ressources ont été supprimées avec ce code de statut.

- 411 Length Required Le serveur a rejeté la requête, car le champ d'en-tête Content-Length n'est pas défini et le serveur l'impose.

- 412 Precondition Failed Le client a indiqué des préconditions dans ses en-têtes que le serveur ne remplit pas.

- 413 Payload Too Large L'entité demandée est plus grosse que la limite définie par le serveur. Le serveur peut fermer la connexion ou retourner un champ d'en-tête Retry-After.

- 414 URI Too Long L'URI interrogé par le client est plus long que ce que le serveur est en mesure d'interpréter.

- 415 Unsupported Media Type Le format média des données demandées n'est pas supporté par le serveur, donc le serveur rejette la requête.

- 416 Range Not Satisfiable La plage spécifiée par le champ d'en-tête Range de la requête ne peut pas être satisfaite ; il est possible que la plage excède la taille des données provenant de l'URI ciblé.

- 417 Expectation Failed Ce code de réponse signifie que les attentes indiquées par le champ d'en-tête de requête Expect n'ont pas pu être satisfaites par le serveur.

- 418 I'm a teapot Le serveur refuse de brasser du café avec une théière.

- 421 Misdirected Request La requête a été envoyée à un serveur incapable de produire une réponse. Ce code peut être envoyé par un serveur qui n'a pas été configuré pour produire des réponses sujettes à la combinaison de schémas et d'identités incluse dans l'URI de la requête.

- 422 Unprocessable Entity (WebDAV) La requête a bien été constituée mais n'a pas pu être traitée à cause d'erreurs sémantiques.

- 423 Locked (WebDAV) La ressource qui est en train d'être consultée est verrouillée.

- 424 Failed Dependency (WebDAV) La requête a échoué à cause de l'échec d'une requête précédente.

- 425 Too Early Indiquer que le serveur ne souhaite pas traiter une requête qui pourrait être rejouée.

- 426 Upgrade Required Le serveur refuse de traiter la requête en utilisant le protocole actuel mais peut accepter de le faire si le client opte pour un autre protocole. Le serveur doit envoyer un en-tête Upgrade (en-US) dans la réponse 426 pour indiquer le(s) protocole(s) demandé(s) (Section 6.7 de [RFC7230]).

- 428 Precondition Required Le serveur d'origine impose que la requête soit conditionnelle. Ceci est prévu pour empêcher le problème de 'perte de mise à jour', où un client récupère l'état d'une ressource avec GET, le modifie, et le renvoie au serveur avec PUT pendant qu'un tiers modifie l'état du serveur, ce qui conduit à un conflit.

- 429 Too Many Requests L'utilisateur a émis trop de requêtes dans un laps de temps donné.

- 431 Request Header Fields Too Large Le serveur n'est pas disposé à traiter la requête, car les champs d'en-tête sont trop longs. La requête peut être renvoyée après avoir réduit la taille des en-têtes.
- 451 Unavailable For Legal Reasons L'utilisateur tente d'accéder à une ressource illégale, telle qu'une page censurée par un gouvernement.

1.5.5 Réponses d'erreur côté serveur

- 500 Internal Server Error Le serveur a rencontré une situation qu'il ne sait pas traiter.
- 501 Not Implemented La méthode de requête n'est pas supportée par le serveur et ne peut pas être traitée. Les seules méthodes que les serveurs sont tenus de prendre en charge (et donc pour lesquelles ils ne peuvent pas renvoyer ce code) sont GET et HEAD.
- 502 Bad Gateway Cette réponse d'erreur signifie que le serveur, alors qu'il fonctionnait en tant que passerelle pour recevoir une réponse nécessaire pour traiter la requête, a reçu une réponse invalide.
- 503 Service Unavailable Le serveur n'est pas prêt pour traiter la requête. Les causes les plus communes sont que le serveur est éteint pour maintenance ou qu'il est surchargé. Notez qu'avec cette réponse, une page ergonomique peut expliquer le problème. Ces réponses doivent être utilisées temporairement et le champ d'en-tête Retry-After doit, dans la mesure du possible, contenir une estimation de l'heure de reprise du service. Le webmestre doit aussi faire attention aux en-têtes de mise en cache qui sont envoyés avec cette réponse (qui ne doivent typiquement pas être mis en cache).
- 504 Gateway Timeout Cette réponse d'erreur est renvoyée lorsque le serveur sert de passerelle et ne peut pas donner de réponse dans les temps.
- 505 HTTP Version Not Supported La version de HTTP utilisée dans la requête n'est pas prise en charge par le serveur.
- 506 Variant Also Negotiates Le serveur a une erreur de configuration interne : la négociation de contenu transparente pour la requête aboutit à une dépendance circulaire.
- 507 Insufficient Storage (WebDAV) Le serveur a une erreur de configuration interne : la ressource sélectionnée est configurée pour participer elle-même à une négociation de contenu transparente, et n'est par conséquent pas un nud terminal valable dans le processus de négociation.

  508 Loop Detected (WebDAV) Le serveur a détecté une boucle infinie en traitant la requête.

510 Not Extended Des extensions supplémentaires sont requises afin que le serveur puisse satisfaire la requête.

511 Network Authentication Required Le code de statut 511 indique que le client doit s'authentifier afin de pouvoir accéder au réseau

1.6 En-têtes HTTP (Header)

Les en-têtes HTTP permettent au client et au serveur de transmettre des informations supplémentaires avec la requête ou la réponse. Un en-tête de requête est constitué de son nom (insensible à la casse) suivi d'un deux-points :, puis de sa valeur (sans saut de ligne). L'espace blanc avant la valeur est ignoré.

Les en-têtes peuvent être groupés selon leur contexte :

- En-tête général : en-têtes s'appliquant à la fois aux requêtes et aux réponses mais sans rapport avec les données éventuellement transmises dans le corps de la requête ou de la réponse.
- En-tête de requête : en-têtes contenant plus d'informations au sujet de la ressource à aller chercher ou à propos du client lui-même.
- En-tête de réponse : en-têtes contenant des informations additionnelles au sujet de la réponse comme son emplacement, ou au sujet du serveur lui-même (nom et version, etc.)
- En-tête d'entité : en-têtes contenant plus d'informations au sujet du corps de l'entité comme la longueur de son contenu ou son type MIME.

La liste suivante résume les en-têtes HTTP en fonction de leur utilisation.

1.6.1 Authentification

- **WWW-Authenticate** définit la méthode d'authentification qui doit être utilisée pour obtenir l'accès à la ressource.
- **Authorization** contient les informations d'identification pour authentifier un agent utilisateur avec un serveur.
- **Proxy-Authenticate (en-US)** définit la méthode d'authentification qui doit être utilisée pour obtenir la ressource derrière un serveur mandataire.
- **Proxy-Authorization (en-US)** contient les informations d'identification nécessaires pour authentifier un agent utilisateur avec un serveur mandataire.

1.6.2 Mise en Cache

- **Age** la durée en secondes passée par l'objet dans un cache proxy.

- **Cache-Control** spécifie des directives pour les mécanismes de mise en cache dans les requêtes et les réponses.
- **Clear-Site-Data (en-US)** nettoie les données de navigation (mouchards (cookies), stockage et cache) associé au site demandé.
- **Expires** la date et l'heure après lesquelles la réponse est considérée comme périmée.
- **Pragma (en-US)** en-tête spécifique à la mise en uvre pouvant avoir divers effets le long de la chaîne de requête-réponse. Utilisé pour la rétrocompatibilité avec les caches HTTP / 1.0 où l'en-tête Cache-Control n'est pas encore présent.
- **Warning (en-US)** un champ d'avertissement général contenant des informations sur les problèmes possibles.

1.6.3 Conditionnels

- **Last-Modified** c'est un validateur, la dernière date de modification de la ressource, utilisée pour comparer plusieurs versions de la même ressource. Il est moins précis que ETag, mais plus facile à calculer dans certains environnements. Les requêtes conditionnelles utilisant If-Modified-Since et If-Unmodified-Since (en-US) utilisent cette valeur pour modifier le comportement de la requête.
- **ETag** c'est un validateur, une chaîne unique identifiant la version de la ressource. Les requêtes conditionnelles utilisant If-Match (en-US) et If-None-Match utilisent cette valeur pour changer le comportement de la requête.
- **If-Match (en-US)** rend la requête conditionnelle et n'applique la méthode que si la ressource stockée correspond à l'un des ETags donnés.
- **If-None-Match** rend la requête conditionnelle et n'applique la méthode que si la ressource stockée ne correspond à aucun des ETags donnés. Ceci est utilisé pour mettre à jour les caches (pour les requêtes sécurisées), ou pour empêcher de télécharger une nouvelle ressource lorsqu'elle existe déjà.
- **If-Modified-Since** rend la requête conditionnelle et attend que l'entité soit transmise seulement si elle a été modifiée après la date donnée. Ceci est utilisé pour transmettre des données uniquement lorsque le cache est obsolète.
- **If-Unmodified-Since (en-US)** rend la demande conditionnelle et attend que l'entité soit transmise seulement si elle n'a pas été modifiée après

la date donnée. Ceci est utilisé pour assurer la cohérence d'un nouveau fragment d'une plage spécifique avec les précédentes, ou pour implémenter un système de contrôle de concurrence optimiste lors de la modification de documents existants.

- **Vary** détermine comment faire correspondre les futurs en-têtes de demande pour décider si une réponse mise en cache peut être utilisée plutôt que d'en demander une nouvelle au serveur d'origine.

1.6.4 Gestion de connexion

- **Connection** contrôle si la connexion réseau reste ouverte après la fin de la transaction en cours.
- **Keep-Alive (en-US)** contrôle la durée pendant laquelle une connexion persistante doit rester ouverte.

1.6.5 Cookies

- **Cookie (en-US)** contient les cookies HTTP stockés précédemment envoyés par le serveur à l'aide de l'en-tête Set-Cookie.
- **Set-Cookie** envoie des cookies du serveur à l'agent utilisateur.

1.6.6 Cross-Origin Resource Sharing (CORS)

- **Access-Control-Allow-Origin** indique si la réponse peut être partagée.
- **Access-Control-Allow-Credentials (en-US)** indique si la réponse à la demande peut être exposée lorsque l'indicateur d'informations d'identification est vrai.
- **Access-Control-Allow-Headers (en-US)** utilisé en réponse à une demande de contrôle en amont pour indiquer quels en-têtes HTTP peuvent être utilisés lors de la requête effective.
- **Access-Control-Allow-Methods** spécifie la ou les méthodes autorisées lors de l'accès à la ressource en réponse à une demande de contrôle en amont.
- **Access-Control-Expose-Headers (en-US)** indique en-têtes pouvant être exposés dans le cadre de la réponse en listant leurs noms.
- **Access-Control-Max-Age (en-US)** indique la durée pendant laquelle les résultats d'une demande de contrôle en amont peuvent être mis en cache.

- **Access-Control-Request-Headers** utilisé lors de l'émission d'une demande de contrôle en amont pour indiquer au serveur les en-têtes HTTP qui seront utilisés lors de la requête effective.

- **Access-Control-Request-Method (en-US)** Utilisé lors de l'émission d'une demande de contrôle en amont pour indiquer au serveur la méthode HTTP à utiliser lors de la requête.

- **Origin** indique l'origine d'une consultation.

- **Timing-Allow-Origin (en-US)** spécifie les origines autorisées à voir les valeurs des attributs récupérés via les fonctionnalités de l'API Resource Timing (en-US), qui seraient autrement signalées comme étant zéro en raison des restrictions d'origines croisées.

1.6.7 Téléchargements

- **Content-Disposition** est un en-tête de réponse si la ressource transmise doit être affichée en ligne (comportement par défaut lorsque l'en-tête n'est pas présent), ou doit être traitée comme un téléchargement et le navigateur doit présenter une fenêtre "Enregistrer sous".

1.6.8 Informations sur le corps du message

- **Content-Length** indique la taille du corps d'entité, en nombre décimal d'octets, envoyée au destinataire.

- **Content-Type** indique le type de média de la ressource.

- **Content-Encoding** utilisé pour spécifier l'algorithme de compression.

- **Content-Language** décrit la (les) langue(s) destinée(s) à l'audience, de sorte qu'elle permette à l'utilisateur de se différencier en fonction de la langue préférée de l'utilisateur.

- **Content-Location (en-US)** indique un emplacement pour les données renvoyées

1.6.9 Informations sur le corps du message

- **Location** indique l'URL de la page de redirection.

1.6.10 Contexte de requête

- **From (en-US)** contient une adresse électronique Internet pour un utilisateur humain qui contrôle l'agent utilisateur demandeur.
- **Host** indique le nom de domaine du serveur (pour l'hébergement virtuel) et (facultativement) le numéro de port TCP sur lequel le serveur écoute.
- **Referer** L'adresse de la page Web précédente à partir de laquelle un lien vers la page actuellement demandée a été suivi.
- **Referrer-Policy** indique quelles informations de provenance envoyées dans l'en-tête Referer doivent être incluses dans les requêtes effectuées.
- **User-Agent (en-US)** contient une chaîne caractéristique qui permet aux homologues du protocole réseau d'identifier le type d'application, le système d'exploitation, le fournisseur du logiciel ou la version du logiciel client.

1.6.11 Contexte de réponse

- **Allow** répertorie l'ensemble des méthodes de requête HTTP prises en charge par une ressource.
- **Server** contient des informations sur le logiciel utilisé par le serveur d'origine pour gérer la demande.

1.7 Protocole HTTPS

1.7.1 Qu'est-ce que le HTTPS?

L'HTTPS, pour Hypertext Transfer Protocol Secure, désigne la version sécurisée du protocole http. En français, le terme se traduit par protocole de transfert hypertexte sécurisé. Il s'agit en réalité de la combinaison entre le langage http et un protocole (SSL ou TLS) de sécurisation des échanges sur le Web. Cette combinaison consiste à protéger l'authentification d'un serveur, la confidentialité et l'intégrité des données échangées et, parfois, l'authentification du client.

HTTPS permet au visiteur de vérifier l'identité du site web auquel il accède, grâce à un certificat d'authentification émis par une autorité tierce, réputée fiable (et faisant généralement partie de la liste blanche des navigateurs internet). Il garantit théoriquement la confidentialité et l'intégrité des données envoyées par l'utilisateur (notamment des informations entrées dans les formulaires) et reçues du serveur. Il peut permettre de valider l'identité du visiteur, si celui-ci utilise également un certificat d'authentification client

1.7.2 Le but du HTTPS

Le HTTPS remplit deux fonctions :

- **Allow** La communication entre le client Web et le serveur Web est chiffrée. Ceci afin dempêcher quun tiers non autorisé ń écoute ż la communication en prenant par exemple connaissance du trafic en réseau WLAN.
- **Server** Le serveur Web est authentifié par le fait quen tout début de communication, un certificat est envoyé au client Web pour attester de la crédibilité du domaine. Cette mesure contribue à combattre les tromperies résultant de faux sites Web.