Devoir Protocole HTTP

HENQUEZ Clément

PSR14687@students.ephec.be

Table des matières

1. Théorie

1.1. Introduction au protocole HTTP

Le protocole HTTP est un protocole de communication client-serveur, inventé par Tim Berners-Lee en 1990. Grâce à ce protocole et le langage HTML il créa le World Wibe Web. HTTP a finalement supplanté le File Transfer Protocol (FTP) du fait que ce dernier ne supportait pas la notion de format des données.

La première version du protocole était très simple. Il avait une seule commande, GET, avec laquelle on pouvait obtenir un fichier HTML. Il n'y avait pas d'en-tête, de codes d'erreur, ou de recours à d'autres types de fichiers, mais il était très petit. Cependant, les limites ont conduit à la sortie du second HTTP/1 en 1995. Il était maintenant possible de connaitre la version du protocole ainsi que l'introduction des codes d'état et des en-têtes. De plus, il était possible d'envoyer d'autres type de contenu que du HTML.

En 1994, apparaît donc le protocole HTTPS qui combine le protocole HTTP ainsi qu'une couche de chiffrement TLS. Un certificat d'authentification émis par une société tierce de confiance permet alors au visiteur de bien vérifier l'identité du site web. De cette façon, il peut confirmer que la page est fiable car les données envoyées et reçues du serveur sont intègres.

Dès janvier 1997, la première version standardisée HTTP/1.1 apparait. Celle-ci ajoute des améliorations majeures : réutilisation des connexions TCP, envoi de plusieurs requêtes successives (pipelining), meilleur gestion du cache, négociation de type de contenu et la présence de l'entête « Host » obligatoire dans les requêtes afin de supporter l'hébergement mutualisé (hébergement de plusieurs sites sur la même adresse IP). Cette version restera la norme pendant plus de vingt ans.

Au fil du temps, les pages web s'alourdissent et deviennent de plus en plus riches. Google conçoit SPDY dans le but d'améliorer les performances. HTTP/2, standardisé en 2015, s'inspire de cette conception. Ce protocole binaire puis multiplexé, qui réduit la latence tout en compressant les en-têtes, permet diverses requêtes simultanées sur une seule connexion. On l'adopte rapidement, surtout sur les grands sites web. En 2022, HTTP/3 s'est imposé comme norme, cette version utilise QUIC, qui mise sur UDP plutôt que TCP. Cela renforce la sûreté des liens et compresse encore davantage les temps de réponse.

Depuis l'invention du Web par Tim Berners-Lee, HTTP n'a jamais cessé d'évoluer : d'un protocole minimaliste limité au texte en 1990, il est devenu une infrastructure puissante, sécurisée et d'une performance exceptionnelle, pilier du Web moderne.

1.2. Interactions client-serveur

Les interactions entre un client (généralement un navigateur web ou une application) et un serveur lors d'une requête HTTP suivent un cycle bien défini, appelé le cycle Requête-Réponse (Request-Response). Ce cycle est composé comme suit :

1. Etablissement de la connexion (TCP)

Le client doit établir une connexion réseau avec le serveur grâce à une résolution DNS (le client utilise le nom de domaine pour trouver l'adresse IP correspondante) puis une connexion TCP/IP entre le serveur à son adresse IP et le port du client (le port standard pour HTTP est 80).

2. Envoi de la requête par le client

Une fois la connexion établie, le client envoie la requête HTTP au serveur. Celle-ci est composée de trois parties qui seront expliquées au point 1.3.

3. Traitement par le serveur

Après avoir reçu le serveur analyse la requête et l'exécute en fonction de la méthode demandée (GET, POST, ...). Il génère ensuite la réponse qu'il enverra au client.

4. Envoie de la réponse par le serveur

Le serveur renvoie la réponse HTTP au client. Elle est également composée de trois parties qui seront expliquées au point 1.6.

5. Fermeture (ou réutilisation) de la connexion

Enfin, le client reçoit la réponse, interprète et affiche le contenu. Si l'en-tête « Connection : keep-alive » est utilisé, la connexion reste ouverte pour de futures requêtes. Ce qui permet d'améliorer mes performances en évitant la première étape de ce cycle.

1.3. Structure d'une requête HTTP

Une requête HTTP est composée de trois parties : la ligne de requête qui est obligatoire et en première ligne, des entêtes optionnels et le corps de requête aussi optionnel.

Premièrement, la request-line (ligne de requête) est définie comme suit :

« METHOD REQUEST-URI HTTP-VERSION »

Les méthodes sont les verbes HTTP qu'on définira plus bas au point 1.9. Les plus courants sont GET, POST, PUT, DELETE. Ensuite, request-URI (Uniform Resource Indicator) est la ressource à appeler. L'URI peut être absolue ou relative mais généralement les relatives sont plus utilisées avec

un header « HOST ». Finalement le champ « HTTP-VERSION » contient la version du protocole HTTP comme par exemple HTTP/1.1.

Ensuite, les headers (entêtes) donnent des informations supplémentaires sur la requête elle-même, le client, le serveur, le corps de la requête (s'il existe), et la façon dont la communication doit être gérée. Les headers standards principaux sont : « Accept », « Accept-Charset », « Accept-Encoding » et « Accept-Language ». Ceux-ci seront expliqués au point 1.7. Les entêtes sont classés par rapport à leur fonction :

- En-têtes concernant le corps
- En-têtes de contrôle du client
- En-têtes de contrôle de la connection et du flux
- En-têtes de mise en cache

Enfin, le body (corps) est utilisé pour transmettre des données supplémentaires au serveur. C'est dans celui-ci que les informations que le client souhaite envoyer au serveur pour qu'il les traite. Comme dit ci-dessus la présence du corps dépend de la méthode utilisée. Le rôle du corps peut varier :

- Envoi de données pour la création ou la mise à jour
- Soumission de formulaires web
- Téléchargement de fichiers

Lorsque le corps est présent des headers additionnels : « Content-Type » et « Content-Length » doivent être précisés pour définir son type et sa taille en octets. Le corps est généralement absent ou ignoré pour les requêtes qui ne nécessitent pas l'envoi de données pour être traitées comme pour « GET », « HEAD » et « DELETE ».

1.4. Structure d'une URL

Une URL (Uniform Resource Locator) est une URI spécifique. Celle-ci est l'adresse complète utilisée pour localiser une ressource sur Internet. Sa structure se compose de plusieurs parties, dont la présence de certaines peuvent être facultative. Prenons comme exemple une URL complète :

http://www.example.com:80/path/to/mypage.html?
key1=value1&key2=value2#somewhereinthedocument

La première partie de l’URL est nommée le schéma, elle indique le protocole que le navigateur doit utiliser pour accéder à la ressource. Le schéma est suivi du séparateur « :// ». Dans notre exemple le schéma est « http ».

La deuxième s’appelle l’autorité. Celle-ci inclut le domaine (ici www.example.com) et le port (80) séparé par un double-point.

La troisième est le chemin de la ressource vers le serveur web. Dans notre exemple : « /path/to/mypage.html ».

La quatrième partie sont les paramètres, ils sont de la forme d’une liste de clés et valeurs chacune séparée par une esperluette. Ces paramètres donnent des informations supplémentaires au serveur pour qu’il effectue des traitements supplémentaires avant d’envoyer sa réponse. Ces paramètres ne sont pas obligatoires et sont séparés du chemin par un point d’interrogation. Dans notre exemple : «?key1=value1&key2=value2 ».

Enfin, la dernière partie est appelée l’ancre, celle-ci pointe vers un emplacement spécifique au sein de la ressource. Celle-ci indique au navigateur d’afficher la ressource situé au niveau de l’ancre. Elle n’est pas obligatoire et est précédée d’un dièse. Dans notre exemple : « #somewhereinthedocument ».

1.5. Fonctionnement d’un formulaire HTML

Les formulaires HTML sont un des vecteurs principaux d'interaction entre un utilisateur et un site web ou une application. Ils permettent à l'utilisateur d'envoyer des données au site web. La plupart du temps, ces données sont envoyées à des serveurs web mais la page peut aussi les intercepter et les utiliser elle-même.

Le formulaire est défini par la balise « <form> » et ses attributs clés :

- action : qui définit l’URL cible du serveur à laquelle les données sont transmises.

- method : qui définit la méthode du protocole HTTP à utiliser.

Une fois que le formulaire a été créé, l’utilisateur le soumet au navigateur pour qu’il crée la requête HTTP. Pour cela, il effectue la collecte des données se trouvant dans les champs du formulaire ayant

un attribut « name ». Il encode ensuite les données en les formatant suivant le type spécifié par l'attribut « enctype ». Enfin il construit la requête en fonction de la méthode.

Après que la requête a été créée, le navigateur l'envoie à l'URL spécifiée par l'attribut action. Enfin le serveur reçoit cette requête et identifie la méthode puis lit et décode les données du corps ou de l'URL pour les utiliser (par exemple, pour créer un compte, enregistrer un commentaire, ou exécuter une recherche).

En bref, le formulaire HTML est la passerelle entre l'interaction humaine sur une page web et les exigences structurées du protocole HTTP.

1.6. Structure d'une réponse

Les réponses HTTP ont un format assez similaire à celui des requêtes. Elles sont aussi composées de trois parties, envoyées dans l'ordre.

1. Ligne de statut (status line)

Elle fournit un résumé du résultat de la requête. Cette ligne est composée elle-même de trois parties. En premier il y a la version du protocole HTTP, puis il y a le code de status. C'est un nombre à trois chiffre qui indique le résultat de la requête. Finalement, il y a la phrase descriptive du code de status. Les status codes seront expliqués au point 1.8. Exemple d'une ligne de status : « HTTP/1.1 404 Not Found ».

2. En-têtes de réponse

Au même titre qu'une requête, une réponse peut contenir des headers. Ils définissent le type et la taille du body de réponse, ou des paramètres pour indiquer au client que la réponse peut être stockée en cache. D'autres headers définissent la ressource à appeler en cas de redirection. Les en-têtes sont définis de la même manière que ceux des requêtes par une paire clé-valeur. Les en-têtes sont séparés de la ligne de status par un saut de ligne.

3. Corps de la réponse

Le corps est la partie de la réponse qui contient la ressource réelle demandée. Il est séparé des en-têtes par une ligne vide. Le contenu de ce corps peut être un code HTML pour une page web, le contenu binaire pour une image ou un objet JSON ou XML pour une API. Il peut aussi y avoir une absence de corps en fonction de ka méthode de requête ou lorsque le code de status indique que rien ne doit être retourné.

1.7. Définition des headers les plus importants

« Host » : Indique le nom de domaine du serveur cible.

« User-Agent » : Identifie le logiciel client (navigateur, application, …).

« Authorization » : Contient les identifiants de l'utilisateur.

« Location » : Utilisé avec les codes de status de redirection, il indique l'URL vers laquelle le client doit se rediriger.

« Content-Type » : Indique le format précis du corps de message (JSON, HTML, JPEG).

« Content-Lenght » : Indique la taille exacte du corps de message en octets.

« Accept » : Indique les formats de contenu que le client accepte en réponse.

« Cache-Control » : Contrôle le comptement de mise en cache.

« If-None-Match » : Renvoie la ressource uniquement si son Etag ne correspond pas à cette valeur.

« ETag » : Identifiant unique pour la version d'une ressource, utilisé avec l'en-tête « If-None-Match ».

« Set-Cookie » : Envoie des données (cookies) du serveur au client, demandant au client de les stocker et de les renvoyer dans les requêtes futures via l'en-tête « Cookie ».

« Cookie » : Contient les cookies précédemment stockés par le client, permettant de maintenir une session ou de personaliser le contenu.

« Strict-Transport-Security » : Force le navigateur à utiliser que le protocole HTTPS pour ce domaine même si l'utilisateur saisit http.

« Keep-Alive » : Contrôle la durée pendant laquelle une connexion persistente devrait rester ouverte.

1.8. Définition des status codes les plus importants

Il existe 5 classes de status codes. Ceux-ci indiquent si une requête HTTP a été exécutée avec succès ou non.

1. les réponses informatives (1xx)

« 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 » de la part du client et indique le protocole sur lequel passe le serveur.

« 102 Processing » : 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.

2. les réponses de succès (2xx)

« 200 OK » : La requête a réussi.

« 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.

« 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.

3. les messages de redirection (3xx)

« 300 Multiple Choices » : La requête a plusieurs réponses possibles. L'agent utilisateur ou l'utilisateur doit choisir l'une d'entre elles.

« 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.µ

« 302 Found » : Ce code de réponse indique que l'URI de la ressource demandée a été modifiée temporairement.

4. les réponses d’erreur côté client (4xx)

« 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 « non-autorisé », la sémantique de cette réponse correspond à « non-authentifié » : le client doit s'authentifier afin d'obtenir la réponse demandée.

« 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.

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

5. les réponses d’erreur côté serveur (5xx)

« 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.

« 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.

1.9. Définitions des verbes les plus importants

« GET » : utilisé lorsque l’utilisateur veut récupérer des informations sans apporter de modifications au serveur. C’est la méthode la plus utilisée pour consulter des données (par exemple, afficher une page web ou rechercher des informations dans une base de données).

« POST » : utilisé pour envoyer des données au serveur lorsque l’utilisateur veut créer une nouvelle ressource (un nouvel utilisateur, article ou produit, etc.) ou traiter des informations. Il est souvent utilisé dans les formulaires web (comme l’inscription d’un utilisateur) ou pour soumettre des données qui ne sont pas récurrentes.

« PUT » : utilisé pour mettre à jour ou remplacer une ressource existante sur le serveur, ou pour en créer une nouvelle si elle n’existe pas encore. Il convient lorsque l’utilisateur sait précisément quelle ressource il veut remplacer ou mettre à jour.

« DELETE » : utilisé pour supprimer une ressource spécifique sur le serveur.

« PATCH » : utilisé lorsque l’utilisateur veut modifier partiellement une ressource sans la remplacer complètement.

« HEAD » : utilisé lorsque l’utilisateur veut obtenir uniquement les en-têtes de réponse HTTP, sans le corps du contenu.

« OPTIONS » : utilisé pour demander quelles méthodes HTTP (GET, POST, etc.) sont supportées sur une ressource spécifique.

« TRACE » : utilisé pour déboguer et retracer le chemin d’une requête HTTP à travers plusieurs proxys ou serveurs intermédiaires. Il affiche la requête telle qu’elle est vue par le serveur.

« CONNECT » : utilisé pour établir un tunnel sécurisé via un proxy, souvent pour des connexions HTTPS.

2. Partie Pratique

2.1 Ecrire une requête HTTP brute

Pour cette partie pratique, je vais écrire une requête HTTP brute depuis le terminal. Pour ce faire, j'utilise l'outil Netcat qui permet de lire et écrire des données sur des connexions réseau. Ma requête HTTP a pour but de faire une recherche du protocole HTTP sur « www.google.com ».

Premièrement afin d'écrire la requête HTTP, il faut établir la connexion au serveur de Google sur le port 80, qui est le port standard pour le protocole HTTP. La commande a rentrer dans le terminal est : « nc www.google.com 80 ».

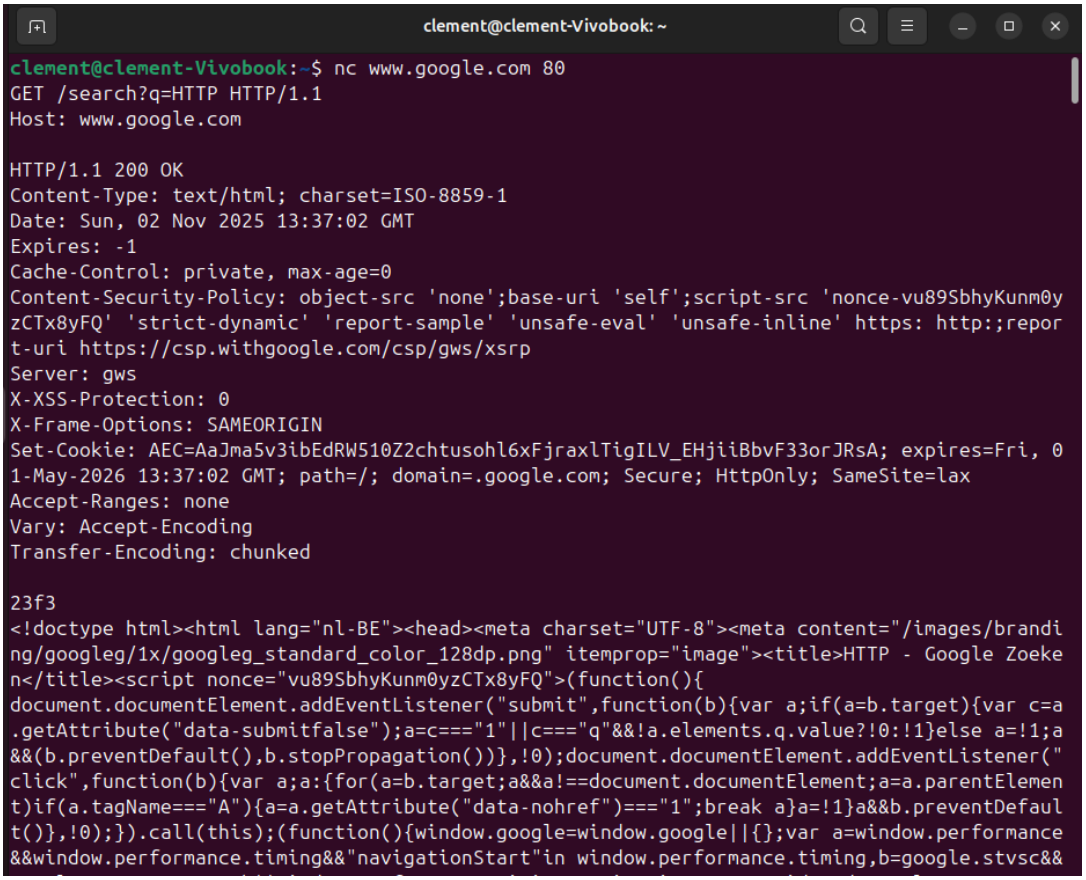
Une fois la connexions établie, j'écris les lignes de la requête HTTP :

GET /search?q=HTTP HTTP/1.1

Host : www.google.com

Pour effectuer une recherche, on utilise la méthode GET car le serveur doit renvoyer les résultats qui correspondent à la recherche qu'on a faite. « /search » est le path et «?q=HTTP » sont les paramètres de requête. HTTP/1.1 indique la version du protocole HTTP à utiliser. L'en-tête Host indique le nom de domaine de la machine à laquelle on veut accéder.

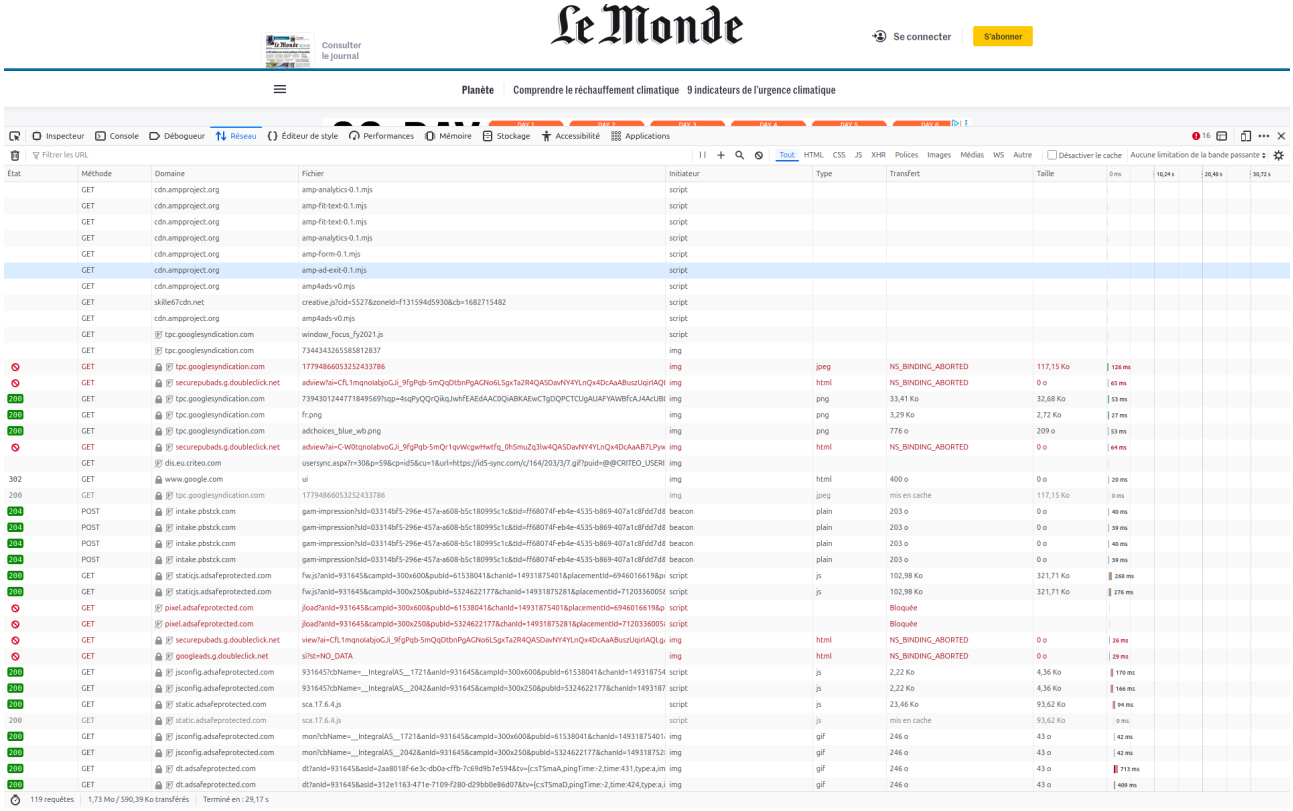
Une fois la requête écrite, on reçoit la réponse du serveur dans notre terminal. Voici la réponse obtenue :



Comme vu dans la partie théorique, la réponse se compose de trois parties : en premier, il y a la ligne de statut « HTTP/1.1 200 OK ». Cela signifie que la requête a été comprise et qu'elle a pu être traitée par le serveur. Ensuite, il ya les en-têtes (headers) qui vont de « Content-Type » à « Transfer-Encoding ». Finalement il y a le corps de la réponse qui commence à partir de « 23f3 » et qui contient la ressource demandée.

2.2. Utiliser l'onglet « Network » de l'outil devtools présent sur le navigateur web pour étudier les requêtes HTTP effectuées par le navigateur, et les réponses envoyées par le serveur.

Afin d'accéder à l'onglet « Network » de l'outil devtools du navigateur web, il suffit de faire un clic droit sur la page internet et de l'inspecter ou d'utiliser le raccourcis clavier Ctrl + Shift + I.



Nous pouvons remarquer les différentes colonnes : état, méthode, domaine, fichier, initiateur, type, transfert, taille et le temps en ms qui donnent des informations sur chaque requête.

En cliquant sur une de ces requêtes, on a plusieurs informations, dont les headers de la requêtes ainsi que de la réponse et les corps de la requête et de la réponse.

Pour mon exemple, j'ai choisi d'inspecter la page d'accueil du site d'information « Le Monde », nous pouvons voir un grand nombre de requêtes permettant de charger plusieurs ressources

différentes (HTML, Javascript, images, publicités). Nous remarquons aussi que certaines de ces requêtes ont abouties alors que d'autres ont échouées. Parmi toutes ces requêtes, certaines ne sont pas destinées au nom de domaine « lemonde.fr » mais à d'autres comme Google, criteo, … Ces requêtes font appel aux services de ces serveurs pour des publicités.

2.3. Ecrire un serveur HTTP basique en Java ou en C sans librairie

Pour cette partie, je vous invite à regarder les autres fichiers du repository Git. J'ai décidé d'écrire le serveur en Java en utilisant aucune librairie exceptée celle de base. Pour ce faire j'utilise un objet ServerSocket de la libraire java.net qui permet d'écouter les connexions sur le port que j'ai spécifié (8080). Une fois le socket créé, il attend une connexion grâce à un bloc try-catch et l'accepte. Puis pour pouvoir lire la requête et écrire la réponse du serveur on a besoin de deux objets, un reader et un writter. Le reader va lire la ligne de la requête, ce qui va permettre de construire la réponse, en ajoutant d'abord la ligne de statut dans le writter. Puis les en-têtes et enfin le corps de la réponse, sans oublier une ligne vide entre les en-têtes et le corps.
En exécutant le serveur basique, un message s'affiche dans la console que le serveur a démarré et lorsqu'on se rend sur http://localhost:8080 on voit s'afficher un message que la requête a bien été traitée et quelle était la requête envoyée au serveur.

Source :

https://fr.wikipedia.org/wiki/Hypertext_Transfer_Protocol

https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/definition-protocole-http/

https://kinsta.com/fr/blog/qu-est-une-requete-http/

https://developer.mozilla.org/fr/docs/Learn_web_development/Extensions/Server-side/First_steps/Client-Server_overview

https://codeka.io/2023/05/25/anatomie-requete-http/

https://developer.mozilla.org/fr/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_URL

https://developer.mozilla.org/fr/docs/Learn_web_development/Extensions/Forms/Your_first_form

https://developer.mozilla.org/fr/docs/Web/HTTP/Reference/Methods

https://sensiolabs.com/fr/blog/2025/les-verbes-http-votre-guide-complet