Le protocole HTTP

Table des matières

Présentation . . . 1
Les commandes . . . 3
- GET . . . 3
- HEAD . . . 4
- POST . . . 4
- PUT . . . 4

Travaux pratiques . . . 4
- Prérequis . . . 4
- Lancement de l'application . . . 4
- Test de l'application . . . 5
- Challenge . . . 5

Techniques de suivi de sessions HTTP . . . 5
- Champs de formulaire cachés . . . 6
- Réécriture de l'URL . . . 6
- Cookies HTTP . . . 7
- API de suivi de session par identifiant de session . . . 8

Présentation

HTTP (Hyper Text Transfert Protocol) a été inventé par Tim Berners-Lee, il définit la façon dont un serveur et un client web échangent des informations. HTTP ne définit ni le format des informations transférées (HTML, GIF, JPEG, ...) ni le mode de construction des adresses Web (URLs) qui font l'objet de documents de normalisation à part.

HTTP définit un ensemble de commandes permettant l'interaction entre un client et un serveur qui supportent le protocole. HTTP est un procole déconnecté ou sans état ce qui signifie qu'il n'assure pas le suivi d'un dialogue entre le client et le serveur. HTTP est définit au-dessus de TCP qui est un protocole connecté et utilise le port TCP 80.

Avec HTTP le client initie le dialogue en envoyant une requête au serveur, le serveut répond au client et coupe la communication (HTTP/1.0).

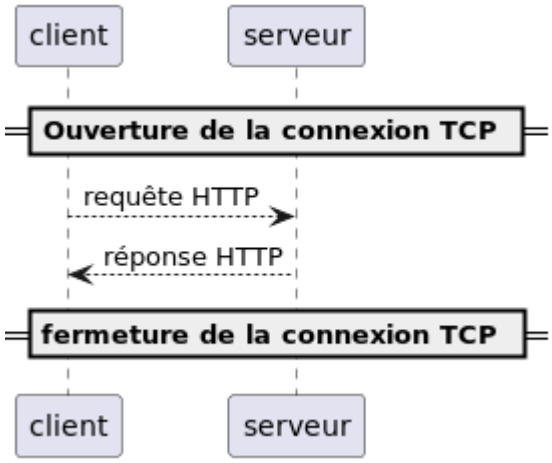

Figure 1. Echange client/serveur HTTP

Listing 1. Structure d'une requête HTTP

```
COMMANDE URI VERSION
En-tête de la requête
éventuellement sur plusieurs lignes
[ligne vide]
Corps de la requête sur plusieurs
lignes si nécessaire
```

Listing 2. Exemple de requête

```
GET / HTTP/1.1
Host: www2.vinci-melun.org
User-Agent: curl/7.54.0
Accept: */*
```

Listing 3. Structure d'une réponse HTTP

```
VERSION CODE-REPONSE TEXTE-REPONSE
entête de réponse
sur plusieurs lignes si
nécessaire
[ligne vide]
corps de la réponse
sur autant de lignes
que nécessaire
```

Listing 4. Exemple de réponse

```
HTTP/1.1 200 OK
Date: Fri, 24 Nov 2017 17:10:31 GMT
Server: Apache/2.4.10 (Debian)
Set-Cookie: PHPSESSID=8rkmkmu65saa92q0b7vig0mg51; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
```

```
Pragma: no-cache
Link: <http://www2.vinci-melun.org/wp-json/>; rel="https://api.w.org/"
Link: <https://wp.me/P8ik5n-ba>; rel=shortlink
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html class="no-js" lang="fr-FR" >
    <head>

....

</html>
```

Les commandes

Les commandes les plus utlisées sont par les navigateurs sont `GET`, `POST` et `HEAD`, les applications REST font également usage de `PUT` et `DELETE`.

GET

Permet de récupérer une ressource depuis une URI sur le serveur. Cette URI peut faire penser à un chemin d'accès à un fichier et c'est souvent le cas côté serveur, mais ce n'est pas une obligation, loin de là

Il est toujours possible de passer des paramêtres au serveur via une requête GET, mais ces paramètres apparaissent dans l'URL. Une URL avec des paramêtres a la forme suivante :

```
http://127.0.0.1/user?id=1274
```

Tous les éléments après le ? sont des paramètres de la requête, le ? lui-même n'en fait pas partie, c'est un délimiteur. Même s'il n'y a pas de norme sur la structure exacte de la requête la forme la plus courante est `clé=valeur` pour un paramètre. S'il y a plusieurs paramètres les paires `clé=valeur` sont séparés par des esperluettes `&`. Par exemple :

```
http://127.0.0.1/chercher?cpost=77000&categorie=5
```

Les caractères autorisés dans la chaîne de requête sont :

- `[A-Z]`
- `[a-z]`
- `[0-9]`
- `*`, `-`, `.` et `_`

- Les autres caractères sont remplacés par leur équivalent numérique hexadécimal précédé du signe `%`, par exemple `@` est codé `%40` et `+` est codé `%2B`

Le résultat d'une requête `GET` ou `HEAD` devrait pouvoir être mis en cache sauf si l'en-tête de réponse `Cache-header` en décide autrement.

HEAD

La commande `HEAD` permet de récupérer l'en-tête correspondant à une ressource la réponse est identique à celle d'une réponse à une commande `GET`, mais sans les données.

POST

Une requête `POST` est utilisée pour envoyer des données au serveur, généralement le contenu d'un formulaire dans le cadre d'une application web traditionnelle. Les données sont stockées dans le corps de la requête et leur type est indiqué par l'entête `content-type` de la requête.

PUT

Une requête `PUT` est utilisée de la même façon qu'une requête `POST`, à un détail prêt : elle est *idempotente*, c'est à dire que peu importe le nombre de fois où elle est envoyée, elle ne modifie l'état du server qu'une seule fois.

Travaux pratiques

Prérequis

- curl opérationnel sur votre système
- une petite application web : sbfirst-session-nom.jar

cURL (abréviation de *client URL request library* : « bibliothèque de requêtes aux URL pour les clients » ou see URL : « voir URL ») est une interface en ligne de commande, écrite en C, destinée à récupérer le contenu d'une ressource accessible par un réseau informatique. https://fr.wikipedia.org/wiki/CURL

Lancement de l'application

```
./sbfirst-session-nom.jar
```

Attendre que l'application se stabilise.

Test de l'application

Plusieurs façons de tester l'application

Listing 5. À l'aide de votre navigateur

```
http://localhost:8080
```

[sbfirst] | *sbfirst8080.png*

Listing 6. En ligne de commande, à l'aide de cURL

```
curl http://localhost:8080
```

Challenge

À l'aide de votre navigateur, tester la route suivante :

```
http://localhost:8080/hello
```

Puis la route suivante, paramétrée :

```
http://localhost:8080/hello?nom=Django
```

Puis, de nouveau la route suivante, sans paramètre :

```
http://localhost:8080/hello
```

Normalement le serveur s'est rappelé de votre dernière requête.

Votre challenge consiste à reproduire ce scénario, mais en ligne de commande, avec cURL. Un rapport détaillé est attendu !

Techniques de suivi de sessions HTTP

Rappel : HTTP est un protocole sans état

Conséquence : un serveur HTTP n'a pas les moyens (via le protocole HTTP) de reconnaître une séquence de requêtes provenant d'un même client. L'adresse IP n'est pas suffisante pour identifier un client parce qu'elle peut faire référence à un serveur proxy sortant par exemple.

Problème : Beaucoup d'applications web doivent gérées des états. Exemple : formulaire multi-pages, gestion d'un caddie, d'une session utilisateur.

Pour contourner ce problème, des données d'état doivent transiter entre les clients et le serveur et être sauvegardées sur un des ces deux tiers. C'est un des aspect très sensible, en terme de cybersécurité, du protocole HTTP.

Voici les solutions les plus courantes pour réaliser un suivi de sessions utilisateur.

Table 1. Techniques de suivi de sessions utilisateur

Solution	Données portées par le client	Données portées par le serveur
Champs cachés de formulaire	x	
Réécriture de l'URL	x	
Cookies persistants	x	
API de suivi de session par identifiant de session	x	x

Champs de formulaire cachés

L'utilisation de champs de formulaire cachés permet au serveur de transmettre des informations au client qu'il retransmet au serveur de façon transparente. Exemple :

```
<form name='formcaddie1' onSubmit='return checkdata()'>
  <input type='hidden' name='code' value='3'>
  <input type='hidden' name='niveau' value='expert'>
  <input type='hidden' name='user' value='julien'>
  ...
  <input type='text' name='quantiteprod' size='3'>
  <p><input type='submit' value='valider' name='valider'>
</form>
```

Avantages

Portabilité : les champs cachés sont supportés par tous les navigateurs.

Inconvénients

- Nécessite toujours une construction dynamique des pages avec toutes les informations.
- Problème côté client lorsque le navigateur s'arrête, qu'une page est mise dans un signet etc. Utilisation systématique de `<form>`

Réécriture de l'URL

L'idée consiste à placer des paramètres dans les URLs renvoyées à l'utilisateur sous forme de liens `<a href='construction dynamique'> lien </a>` afin d'assurer le suivi de session.

Par exemple :

```
<a href='/listerProduits?user=julien&categorie=classique'> lien </a>
```

Avantages

- Portabilité, car supporté par tous les navigateurs.
- Anonymat, l'utilisateur ne se fera connaître qu'au moment de la vente
- N'est pas tenu d'utiliser la balise <FORM>

Inconvénients

- Nécessite toujours une construction dynamique des URLs. Peut être fastidieux à mettre en place et maintenir.
- Les URLs doivent être encodées (présence de caractères réservés)

Cookies HTTP

Intégré au protocole HTTP, un cookie est un ensemble de données texte enregistré sur le poste client qui contient des informations initialement transmises par un serveur web à un navigateur.

Un cookie à une portée (domaine) et une durée de vie. Voir cookie

Lorsqu'un navigateur reçoit en entête HTTP l'instruction (`Set-Cookie: nom=valeur`), le couple clé=valeur est sauvé sur disque et le renvoie systématiquement, à chaque nouvelle requête HTTP du client (dans l'entête HTTP `Cookie: nom=valeur`), au serveur (l'application web) à l'origine du cookie.

```
GET / HTTP/1.1
Host: www.exemple.org
...

HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
...

GET /page.html HTTP/1.1
Host: www.exemple.org
Cookie: name=value
...
```

Avantages

- Facilité et cohérence de mise en œuvre (le client détient les informations le concernant).
- Bonne capacité de personnalisation

Inconvénients

- Pas plus de 20 cookies par domaine et pas plus de 300 cookies par utilisateur
- La taille d'un cookie peut être limitée à 4096 octets (4ko)

API de suivi de session par identifiant de session

Le dispositif de suivi de session est un mécanisme automatique, déclenché coté serveur ; un **identifiant de session** est alors généré qui sera stocké à la fois sur le serveur **et** sur le client, ce dernier aura la responsabilité de le transmettre au serveur lors de chacune de ses requêtes à ce même serveur (par cookie ou paramètre d'url).

L'identifiant de session est une valeur arbitraire qui permet d'identifier (côté serveur) un utilisateur d'un autre. L'identifiant de session fait alors office de clé d'accès à des informations personnelles à une session (un utilisateur). Ces informations sont stockées sur le serveur (en mémoire, fichiers texte ou dans une base de données)

La création de ce type d'identifiant et la technique de transmission (par url ou par cookie) sont pris en charge par les outils de développement web (sous la forme d'une librairie ou d'une classe `Session`). Le développeur ne manipule que rarement l'id de session; l'accès aux données de session s'opère le plus souvent via une structure de type dictionnaire (ensemble de couples clé/valeur).

Avantages

- Traitement générique déjà implémenté
- Utilise la technologie Cookie HTTP

Inconvénients

- Utilise la technologie Cookie HTTP