Interactions client-serveur sur le Web

Le Web est né comme une collection de documents hypertextes accessibles sur Internet. Mais il est devenu bien plus que cela. De nos jours, les sites Web sont de véritables applications, avec une interface graphique (décrite en HTML et CSS), du code client de gestion d'éléments interactifs s'exécutant dans le navigateur (JavaScript) et des traitements de données complexes effectués sur le serveur Web hébergeant le site. Nous abordons dans ce chapitre les éléments communs à la conception d'une application Web interactive : le protocole HTTP et la notion de formulaire Web avant d'aborder les concepts fondamentaux des sites dynamiques.

I Requêtes HTTP(S)

I.1 Décortiquer une URL

La première étape lorsque l'on souhaite accéder à un site Web est la saisie de l'adresse de ce site dans la barre d'adresse du navigateur. Une telle adresse s'appelle une URL (pour l'anglais *Uniform Resource Locator* ou « localiseur uniforme de ressources »). La syntaxe (simplifiée) d'une URL est la suivante :

`protocole://nom-ou-adresse/document`

Le protocole peut être `http` ou `https`. La partie `nom-ou-adresse` peut être le nom de domaine ou encore l'adresse IP de la machine faisant office de serveur. En première approximation, le `document` permet de localiser une ressource (en particulier un fichier) stocké sur le serveur et que le programme client (i.e., le navigateur Web de la personne qui visite le site) souhaite récupérer ou afficher.

I.2 Analyse d'une requête HTTP

I.2.1 Vu de l'extérieur

Par exemple, si l'utilisatrice Alice saisi dans la barre d'adresse de son navigateur l'URL

`http://requetes.infobrisson.fr/index.html`

les actions suivantes se produisent :

1) Le navigateur Web isole le nom de machine `requetes.infobrisson.fr`.
2) Le navigateur Web effectue une requête DNS pour obtenir l'adresse IP du serveur Web hébergeant le site (ici `109.234.162.45`).
3) Le navigateur Web se connecte à la machine dont l'adresse est IP est `109.234.162.45`, en utilisant le protocole TCP, sur le port 80.
4) Une fois la connexion établie, le navigateur Web envoie un certain nombre de messages en se conformant au protocole HTTP, pour demander la ressource `index.html`.
5) Le serveur Web se trouvant à l'adresse `109.234.162.45` envoie en réponse le contenu du fichier au navigateur Web d'Alice.
6) Le navigateur Web d'Alice peut ensuite parcourir le fichier, et afficher la page correspondante.


Info

Le processus ci-dessus est répété à chaque fois que la navigation d'Alice nécessite une nouvelle ressource : lorsqu'elle clique sur un lien, lorsque la page contient une image ou une vidéo, lorsqu'elle soumet des informations au moyen d'un formulaire.

Définition

Le protocole HTTP (pour l'anglais *HyperText Transfert Protocol* soit « protocole de transfert hypertexte ») est un protocole de type client-serveur. Le serveur Web est le programme spécialisé chargé d'attendre des messages via des connexions réseaux TCP et d'y répondre. Par extension, on appelle aussi *serveur Web* la machine connectée au réseau qui exécute ce programme. Les *clients* sont les navigateurs Web se connectant aux sites Web et par extension les machines sur lesquelles ces navigateurs s'exécutent. Un *site Web* est un ensemble de ressources (en particulier des fichiers au format HTML) associées à une URL et distribuées par un serveur Web.

I.2.2 Contenu d'une requête

Le protocole HTTP est un standard défini par l'IETF, le même organisme qui standardise IP et TCP. Ce protocole définit les messages envoyés entre le navigateur et le serveur Web. Les messages envoyés par le client sont appelés des *requêtes*. Ceux envoyés par le serveur sont appelés des *réponses*. La majorité des requêtes faites par le client sont des demandes d'une ressource (comme un fichier HTML). Par exemple, en navigant vers l'URL `http://requetes.infobrisson.fr/index.html`, le navigateur Web envoie au serveur la requête :

```
GET /index.html HTTP/1.1
Host: requetes.infobrisson.fr
```

Par cette requête, le client annonce au serveur `requetes.infobrisson.fr` qu'il veut communiquer en utilisant la version `1.1` du protocole HTTP et lui demande la ressource `/index.html`. La réponse du serveur est alors :

```
HTTP/1.1 200 OK
Date: Thu, 02 Apr 2020 08:04:23 GMT
Content-Type: text/html
Content-Length: 251
Last-Modified: Thu, 02 Apr 2020 06:04:07 GMT

<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Page de test</title>
  </head>
  <body>
    <h1>Page de test</h1>
    <p>
      Cette page illustre le fonctionnement du protocole <i>HTTP</i>.
    </p>
  </body>
</html>
```

Par cette réponse, le serveur informe le client :

- qu'il accepte de communiquer avec la version 1.1 du protocole HTTP ;
- que la ressource demandée est disponible et donc que la requête peut être satisfaite (`200 OK` étant le code signifiant le succès de la requête) ;

- de la date de réception de la requête ;
- que le type du fichier est HTML ;
- que le fichier fait exactement 251 octets ;
- de la date de dernière modification du fichier.

Ces informations constituent les *entêtes* de la réponse HTTP. Ces entêtes permettent au client de gérer au mieux la réponse. Par exemple, la taille permet au navigateur de contrôler qu'il a bien reçu l'intégralité du fichier. Le type permet au navigateur de choisir entre afficher le contenu du fichier (éventuellement mis en forme, comme dans le cas d'un fichier au format HTML) ou de le télécharger (par exemple pour un fichier d'archive de type ZIP). Ces entêtes sont suivis d'une ligne vide, puis du contenu du fichier (du texte contenant du code HTML). On peut constater que du « < » initial au « > » final, il y a bien 251 caractères.


Astuce

On eut facilement retrouver ces informations avec les « Outils de développement » des navigateurs modernes.

Si on répète la même expérience en utilisant cette fois la requête :

```
GET /nimportequoi.html HTTP/1.1
Host: requetes.infobrisson.fr
```

alors le serveur répond :

```
HTTP/1.1 404 Not Found
Date: Thu, 02 Apr 2020 08:13:25 GMT
Content-Type: text/html; charset=iso-8859-1
Content-Length: 315

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
<p>Additionally, a 404 Not Found
error was encountered while trying to use an ErrorDocument to handle the request.</p>
</body></html>
```

Ici, le code de la réponse est 404, signifiant que le serveur n'a pas trouvé la ressource demandée (le fichier n'existe pas). Le contenu de la réponse est une petite page HTML décrivant l'erreur. Le protocole prévoit plusieurs codes d'erreurs. Outre le célèbre 404 (ressource indisponible), on rencontre aussi le code 403 (permission refusée) lorsque l'on tente d'accéder à une ressource sans avoir les droits suffisants, ou le code 500 lorsque le serveur rencontre une erreur interne.

Le module `requests` de Python permet de faire des requêtes HTTP :

```python
import requests

req = requests.get("http://requetes.infobrisson.fr/index.html")
print(req.status_code)
print(req.headers)
print(req.text)
```

I.3 Pas de stress, il y a le « S »

Un inconvénient du protocole HTTP est qu'il s'agit d'un protocole « en clair ». N'importe qui ayant les droits suffisants pour intercepter les paquets réseaux échangés entre le client et le serveur peut lire le contenu des messages échangés. Pour ce faire, il suffit d'avoir les droits administrateurs sur l'une des machines se trouvant sur le chemin entre le client et le site Web visité (par exemple sur l'une des passerelles transmettant les paquets réseaux).

Cette caractéristique a des conséquences importantes en termes de sécurité et de confidentialité des données. En particulier, si un utilisateur renseigne dans le formulaire d'un site accessible en HTTP un identifiant et un mot de passe, ces derniers sont facilement récupérables par une tierce partie. Pour remédier à ce problème, le protocole HTTPS (pour l'anglais *Hyper Text Transfer Protocol Secure*, en français « protocole de transfert hypertexte sécurisé ») a été introduit. Pour simplifier, lorsque le client et le serveur utilisent ce protocole, ils chiffrent [1] les messages avant de se les transmettre. Ainsi, un utilisateur interceptant les paquets TCP ne pourra en extraire qu'une suite d'octets d'apparence aléatoire.

II Passage de paramètres et formulaires

II.1 Passage de paramètres

La syntaxe des URL est un peu plus complexe que celle montrée précédemment. Une URL peut être de la forme suivante :

`protocole://nom-ou-adresse:port/document?n1=v1&...&nk=vk#id`

Dans une URL, il est possible de spécifier le port TCP sur lequel effectuer la connexion (par exemple, si l'on exécute plusieurs serveurs Web sur la même machine, ils doivent tous être en écoute sur des ports différents). Le document est écrit comme chemin relatif à partir du répertoire où est stocké le site sur le serveur. Attention, cependant, le document et en général l'URL peut aussi être réécrit par le serveur et transformé en un chemin n'ayant rien à voir. Un exemple courant est celui des pages Web personnelles. On peut par exemple configurer un serveur Web pour que des URL comme « `https://www.mon-site.fr/perso/alice` » correspondent au chemin « `/home/alice/public_html/` ». Il n'y a donc pas réellement de répertoire appelé « perso » sur le serveur.

La partie située après le document et marquée par un « `?` » est la liste des *paramètres de requête*. Ces paramètres sont des paires $\texttt{n}_\texttt{i}\texttt{=v}_\texttt{i}$ où $\texttt{n}_\texttt{i}$ est le nom du paramètre et $\texttt{v}_\texttt{i}$ sa valeur. Les paires sont séparées par un symbole « `&` »». Ces paramètres ne font pas partie du nom de la ressource que l'on essaye de récupérer, mais sont une façon pour le client de passer des valeurs au serveur. Enfin, la portion finale `#id` est appelée un signet et permet de cibler un élément particulier dans la ressource demandée. En pratique, il s'agit de l'identifiant d'un élément HTML de la page demandée.

Activité : Passage de paramètres à un serveur. On illustre le passage de paramètres en utilisant la version française de l'encyclopédie en ligne Wikipédia (`https://fr.wikipedia.org`). En navigant sur cette page, on remarque un champ de recherche (en haut à droite). Lorsque l'on saisit une chaîne de caractères dans le champ de recherche, par exemple « Informatique », on est redirigé vers la page française définissant le terme. Cependant, on peut obtenir le même résultat en écrivant directement ceci dans la barre d'adresse de son navigateur l'URL :

`https://fr.wikipedia.org/w/index.php?search=Informatique`

Dans cette URL, le protocole est `https`, le nom du site est `fr.wikipedia.org`, le document est `/w/index.php`. On passe en plus au serveur le paramètre de requête search avec la valeur `Informatique`. Cela a pour effet de charger la page correspondante. De même, si on effectue la recherche sur un autre mot

`https://fr.wikipedia.org/w/index.php?search=Ordinateur`

alors la page correspondant à ce mot est chargée. Enfin, si on recherche un terme inconnu comme

`https://fr.wikipedia.org/w/index.php?search=Informatik`

1. Le terme de « chiffrement » et le verbe associé « chiffrer » un message sont les termes proposés par l'Agence nationale de la sécurité des systèmes d'information (ANSSI), qui considère « cryptage » et « crypter » comme incorrects.

alors on est redirigé sur une page « spéciale » indiquant qu'aucune page de l'encyclopédie ne correspond exactement au terme recherché et proposant des suggestions. On voit donc que le site effectue un traitement différencié en fonction des divers paramètres passés. En ce qui concerne les signets, on peut naviguer vers l'URL

`https://fr.wikipedia.org/wiki/Informatique#Algorithmique`

et constater que la page chargée défile automatiquement pour afficher la section « Algorithmique ». La portion de code HTML correspondant est la suivante :

```
<h3>
  <span id="Algorithmique">Algorithmique</span>
</h3>
```

Le signet permet donc de faire en sorte qu'une URL pointe non seulement sur un fichier, mais aussi sur un élément HTML particulier de ce fichier.

II.2 Les formulaires

Comme on l'a constaté avec l'activité précédente, le passage de para- mètres sur une page Web repose sur deux ingrédients. Le premier est la capacité pour le serveur de traiter certaines URL de manière spéciale. La seconde est la présence dans la page Web d'éléments graphiques permettant à l'utilisateur de saisir des valeurs. Ces widgets permettant la saisie d'information sont regroupés au sein de formulaires dont nous décrivons maintenant l'utilisation.

En HTML, l'élément `form` permet de regrouper un ensemble d'éléments de saisie dont les valeurs seront envoyées au serveur lors de la soumission du formulaire. La structure usuelle d'un formulaire est la suivante

```
<form action="reponse_formulaire.php">
  Nom : <input type="text" name="nom" />
  Prénom : <input type="text" name="prenom" />
  Age : <input type="text" name="age" />
  <button type="submit">Envoyer</button>
</form>
```

Dans ce fragment de document, on remarque en premier lieu un élément `form` possédant un attribut action. La valeur de ce dernier est l'URL à laquelle envoyer les paramètres. Ces derniers sont à saisir dans trois champs de saisie de texte, correspondant à des éléments HTML `input`. Ces éléments possèdent deux attributs. Le premier, `type`, indique que l'on souhaite un champ de saisie de texte (il existe d'autres types d'éléments). Le second, `name`, indique le nom du paramètre auquel ce champ correspond. Enfin, le dernier élément, `button`, correspond au bouton de soumission du formulaire. L'attribut type vaut `"submit"` et le contenu des balises `<button>...</button>` permet de définir l'étiquette du bouton. Si on saisit trois valeurs, par exemple « Lovelace », « Ada » et « 36 », et qu'on clique sur le bouton, alors le navigateur charge la page [2]

`https://nsi.infobrisson.fr/reponse_formulaire.php?nom=Lovelace&prenom=Ada&age=36`

et affiche une petite liste non énumérée donnant les valeurs des paramètres.

- La valeur du nom est Lovelace
- La valeur du prénom est Ada
- La valeur de l'âge est 36

Le processus de soumission d'un formulaire se décompose en plusieurs étapes.

1) Pour chaque élément graphique dont la valeur est v_i et le nom (donné par l'attribut `name`) est n_i, le navigateur forme la chaîne de caractères n_i=v_i, puis il rassemble tous les éléments entre eux en les séparant par des caractères « `&` ».
2) Le navigateur Web contacte le serveur se situant à l'URL donnée dans l'attribut action de l'élément `form`.

2. Vous pouvez retrouver ceci ici : `https://nsi.infobrisson.fr/form.html`.

3) Le navigateur envoie la chaîne de caractères `n1=v1&...&nk=vk` au serveur. Sans option particulière, cette chaîne est passée comme un paramètre de requête (i.e., ajoutée en fin d'URL avec un « `?` »).
4) Le serveur calcule alors une réponse qui est renvoyée au navigateur.

II.3 Mode de passage des paramètres : les méthodes `GET` et `POST`

Le mode de passage des paramètres par un formulaire est défini par l'attribut method de la balise `<form>`. Si cet attribut est absent (comme dans nos exemples précédents) ou que sa valeur est `get`, alors les paramètres du formulaire sont passés au serveur via l'URL, en suivant la syntaxe

`protocole://nom-ou-adresse:port/document?n1=v1&...&nk=vk`

Si l'attribut method vaut `post`, alors les paramètres sont passés de manière différente. Le navigateur forme bien la chaîne de caractères `n1=v1&...&nk=vk`. Cependant, il envoie un message HTTP commençant par `POST`. Les paramètres sont alors placés dans le *corps de la requête HTTP*. Considérons le formulaire déjà utilisé dans la section précédente.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Un exemple de formulaire</title>
  </head>
  <body>
    <form method="m" action="reponse_formulaire.php">
      Nom : <input type="text" name="nom" />
      Prénom : <input type="text" name="prenom" />
      Age : <input type="text" name="age" />
      <button type="submit">Envoyer</button>
    </form>
  </body>
</html>
```

Si `m` vaut `get`, alors la requête HTTP envoyée par le navigateur au serveur Web est la suivante :

```
GET /reponse_formulaire.php?nom=Lovelace&prenom=Ada&age=36 HTTP/1.1
Host: nsi.infobrisson.fr
```

Comme on le voit, les paramètres font simplement partie de l'URL et donc de la ressource demandée au serveur (première ligne). Si `m` vaut `post`, alors la requête envoyée par le navigateur Web est la suivante :

```
POST /reponse_formulaire.php HTTP/1.1
Host: nsi.infobrisson.fr
Content-Type: application/x-www-form-urlencoded
Content-Length: 30


nom=Lovelace&prenom=-Ada&age=36
```

Ici, la ressource demandée est simplement `/reponse_formulaire.php`. Elle est cependant demandée par une requête HTTP utilisant la méthode POST. Cette méthode permet d'envoyer un contenu long à un serveur. Elle réutilise les mêmes entêtes que pour une réponse du serveur, à savoir `Content-Type` qui indique le type du corps du message envoyé (ici `application/x-www-form-urlencoded` pour indiquer un passage de paramètres utilisant la syntaxe d"URL) et la longueur du contenu en octets. Les deux méthodes de passage de paramètres sont complémentaires et permettent de couvrir des usages différents. Nous détaillons ces différents usages.

II.3.1 Méthode `GET`

L'intérêt de la méthode GET est que toute l'information nécessaire au serveur est contenue dans l'URL. Il est donc possible de mémoriser uniquement cette URL pour s'en servir plus tard. Considérons le site `https://www.data.gouv.fr/`. Ce dernier est la plate-forme des données publiques françaises, sur lequel l'état, les collectivités et les organismes publics déposent des fichiers de données. Comme pour Wikipédia, ce site dispose d'un formulaire de recherche. Si on souhaite trouver toutes les données en rapport avec l'informatique, il est possible d'utiliser directement l'URL suivante :

`https://www.data.gouv.fr/fr/search/?q=informatique`

En particulier, on peut demander au navigateur Web de la sauvegarder comme un « marque-page ». Il est également possible de créer directement un lien vers cette URL dans une page Web.

```
<a href="https://www.data. gouv.fr/fr/search/?q=informatique">
  Liens vers les jeux de données en rapport avec l'informatique
</a>
```

Un tel lien pointera vers la page de résultats, même si de nouvelles entrées sont ajoutées sur le site de destination (la page de résultat étant recalculée à la demande par le serveur). Le protocole HTTP impose aussi que cette méthode ne soit utilisée par les serveurs que pour effectuer des opérations

« sans effet » (ie., qui ne vont pas provoquer de mise à jour ou de changement de l'état interne du serveur). Il est donc possible de recharger plusieurs fois la même URL sans problème.

Il existe des situations où passer des paramètres via l'URL peut poser problème. En premier lieu, les URL ne peuvent pas être de taille arbitrairement longue. Les serveurs Web et les navigateur Web imposent tous des limites à la taille des URL. Ces limites varient selon les programmes mais sont de l'ordre de quelques milliers d'octets. Considérons un formulaire qui permet de saisir du texte de longueur arbitraire.

```
<form action="https://monsite.fr" method="get">
  Vous pouvez écrire un roman ci-dessous
  <textarea name="montexte">

  </textarea>
  <button type="submit">Soumettre son roman !</button>
</form>
```

Si le contenu du champ de saisie est trop long, alors en recevant la requête HTTP le serveur peut renvoyer un code d'erreur 414 (URL too long). Un autre aspect à prendre en considération est la confidentialité. En effet, considérons le formulaire suivant de saisie de mot de passe :

```
<form action="https://monsite.fr" method="get">
  Identifiant : <input type="text" name="ident" />
  Mot de passe : <input type="password" name="pass" />
  <button type="submit">Se connecter</button>
</form>
```

Le mot de passe est masqué et affiché dans la zone de saisie comme une suite d'étoiles (ou de ronds). Cependant, lorsqu'on clique sur le bouton, le navigateur exécute la requête et affiche dans sa barre d'adresse l'URL suivante :

`https://monsite.fr/?ident=toto&pass=g23a1FxA`

Le mot de passe apparaît donc en clair, et une personne présente à proximité de l'écran pourrait l'apercevoir (le rôle des petits ronds affichés à la place du mot de passe étant justement de ne pas permettre à une tierce personne de lire le mot de passe sur l'écran).

II.3.2 Méthode `POST`

Les caractéristiques de la méthode POST sont exactement inverses. Lors du processus de soumission, l'URL n'est pas suffisante : il faut que les paramètres soient passés dans le corps de la requête. Cela signifie en particulier qu'il n'est pas possible de sauvegarder uniquement l'URL comme dans le cas d'une requête de

type GET. Le protocole HTTP demande l'utilisation de la méthode POST pour toute requête effectuant des mises à jour côté serveur. Si on imagine un site de réservation de billets de train, le formulaire permettant de choisir et acheter le billet doit utiliser la méthode POST, car la soumission du formulaire provoquera une mise jour (le billet ne sera plus disponible pour les autres personnes car il aura été vendu). Cette notion de mise à jour implique que recharger une page contenant un formulaire de type POST n'est pas anodin. En poursuivant avec notre exemple d'achat de billet, Supposons que l'on clique sur le bouton de soumission pour finaliser l'achat du billet de train. Si un problème de connexion réseau survient, la page de confirmation peut ne pas se charger. Le navigateur continue d'attendre le résultat (et l'indique visuellement avec un curseur tournant par exemple). On pourrait alors être tenté de recharger la page. Cela impliquerait de renvoyer une nouvelle fois les données du formulaire et d'exécuter une nouvelle fois la mise à jour. Dans notre cas cela pourrait correspondre à l'achat d'un deuxième billet. Les navigateurs Web préviennent l'utilisateur dans ce cas. Par exemple, le navigateur Firefox affichera le message suivant :

```
Pour afficher cette page, les informations précédemment transmises par
Firefox doivent être renvoyées. Ceci répétera toute action (telle qu'une
recherche ou un ordre d'achat) entreprise précédemment
```

Il y à parfois des situations où l'utilisation de la méthode POST est obligatoire. Comme on l'a vu, si la taille des paramètres envoyés est trop importante (par exemple, long champ de saisie de texte ou dépôt d'un fichier sur le serveur) alors seule la méthode POST garantit que la requête ne sera pas tronquée. De même, pour les formulaires demandant un mot de passe, la méthode POST ne rendra pas visible ce dernier dans la barre d'adresse. Attention, l'utilisation de la méthode POST pour les formulaires de mots de passe ainsi que l'utilisation d'un élément de saisie de type password ne sont pas suffisants pour garantir la confidentialité du mot de passe. En effet, ils ne protègent que d'un espionnage direct du mot de passe par une personne présente au moment de la saisie. Une autre mesure indispensable est l'utilisation du protocole HTTPS (et non pas HTTP) pour les échanges entre le client et le serveur. N'importe quelle personne disposant des droits suffisants sur l'une de ces machines intermédiaires peut inspecter le contenu de ces paquets et ainsi connaître le mot de passe envoyé au serveur. Afin de protéger les utilisateurs non avertis, les navigateurs Web modernes affichent un message d'avertissement lorsqu'une page contient un élément input de type password mais que l'URL correspondante utilise le protocole HTTP (non sécurisé).


Info

Loin d'être un simple protocole d'échange de fichiers, le protocole HTTP permet à des pages Web d'envoyer des paramètres de requêtes. Ces derniers sont utilisés pour calculer le contenu de la page qui est renvoyée au client. Le langage HTML permet de décrire des formulaires permettant à l'utilisateur de saisir des valeurs et de les soumettre à un serveur Web. Il existe deux méthodes de passage de paramètres en HTTP, GET et POST. La méthode GET stocke les paramètres dans l'URL elle-même, alors que la méthode POST stocke ces paramètres dans le corps de la requête.

III Sites dynamiques

Nous avons vu qu'un navigateur Web peut envoyer à un serveur des requêtes paramétrées. Il existe deux modes de passage de paramètres, la méthode GET et la méthode POST. Nous faisons maintenant une présentation de deux concepts fondamentaux de la programmation Web côté serveur. Ces concepts sont présents dans tous les langages de programmation Web et en particulier en PHP, dont nous ferons une présentation rapide, orientée par des exemples. Ce chapitre se veut une introduction à la programmation Web côté serveur et n'a pas pour but d'être un cours de programmation en PHP. En effet, l'écriture de traitements complexes côté serveur de même qu'une expertise en PHP est hors programme.

III.1 Concepts fondamentaux du Web côté serveur

III.1.1 Étude de cas et principe de fonctionnement

Considérons un exemple concret. L'utilisatrice Alice navigue vers son site Web favori (un forum de discussion). Arrivée sur la page d'accueil, elle remplit un formulaire avec son identifiant et son mot de passe.

Une fois authentifiée elle peut poster des messages et accéder librement aux parties du site à accès restreint. Quand elle a terminé, Alice clique sur un bouton de déconnexion du site, elle se retrouve donc sur la page d'accueil. Les pages à accès restreint ne sont plus accessibles. Dans le même temps, sur une autre machine cliente, Bob se connecte lui aussi au site et s'identifie. Il utilise le site comme Alice et en même temps qu'elle. Leur vue du site n'est pas mélangée pour autant, le serveur répond à chacune des requêtes en envoyant le contenu attendu par chacun des clients.

Mettons un instant de côté notre exemple, et regardons ce qui se passe côté serveur. La plupart des langages de programmation Web fonctionne de la manière suivante :

1) Le serveur Web reçoit une requête HTTP demandant une certaine ressource (par exemple `index.php?x=1&y=2`).
2) Le serveur détermine que la ressource ne doit pas être renvoyée tel quelle au client mais « évaluée ». Une manière de déterminer cela peut être l'extension du fichier (ici .php), mais ce n'est pas la seule manière de faire.
3) Le serveur Web appelle un programme externe (par exemple l'interprète du langage PHP) pour évaluer le fichier. Il met à disposition de ce programme les paramètres de la requête ainsi que toutes les informations qui sont liées à cette dernière.
4) Le programme effectue un traitement en utilisant ces paramètres.
5) Le programme écrit (dans un fichier de sortie ou sur sa sortie standard) le contenu de la réponse de la requête. En général c'est un document HTML, mais on peut aussi générer du CSV, du PDF, etc.
6) Le serveur Web prend le fichier généré et le renvoie alors comme réponse au navigateur qui à fait la requête.

Dans ce modèle d'exécution, un programme est exécuté côté serveur pour chaque requête HTTP. Il produit une sortie (le résultat de la requête) puis se termine. Cette description ne permet pas de rendre compte de deux phénomènes que l'on a constatés dans notre exemple.

1) Le site peut faire la différence entre Alice et Bob. Plus que ça, il peut personnaliser ses réponses en fonction de l'identité du client.
2) Plus important, entre deux requêtes HTTP distinctes, le serveur se souvient de « l'état » dans lequel était un client particulier lors de la dernière requête. Par exemple tant qu'Alice ou Bob ne se sont pas déconnectés, le serveur se « souvient » qu'il se sont authentifiés un peu plus tôt et donc qu'ils peuvent accéder aux pages restreintes.

Nous allons voir deux ingrédients qui permettent d'obtenir ces fonctionnalités.

III.1.2 Les cookies

Il est important pour un site Web de pouvoir identifier ces différents clients (et les différencier). Cependant, il n'existe rien permettant d'identifier un client de manière unique au niveau réseau. Par exemple, l'adresse IP d'un client, bien qu'unique à un moment donnée n'est pas un identifiant valide.

Par exemple plusieurs personnes, au sein d'un réseau local privé, peuvent partager la même IP publique (celle que le serveur voit). Une adresse IP peut aussi changer. Par exemple si un client est sur un réseau mobile, il peut être coupé du réseau momentanément à cause d'un obstacle, puis reconnecté. Il est alors possible que son adresse IP ait changé.

Une manière pour les sites Web de différencier leurs clients est l'utilisation de cookies. Un cookie est une petite quantité de données. Il est composé d'un nom, d'une valeur et optionnellement d'une date d'expiration. Le nom, la valeur et la durée de vie sont choisis par le serveur. Lorsqu'un serveur veut déposer un cookie particulier chez un client, il l'envoie comme un entête de réponse HTTP. Par exemple, si un client demande une page `index.php`, le serveur peut répondre

```
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Mon, 29 May 2019 17:24:32 GMT
Content-Type: text/html
Content-Length: 1023
Set-Cookie: test=42; Expires=-Mon, 30 May 2020 18:10:12 GMT;
```

En recevant cette réponse du serveur, le navigateur Web voit que le serveur lui envoie un cookie, dont le nom est test, la valeur est 42 et la date d'expiration est le 30 mai 2020 à 18h10, 12 secondes. Le navigateur stocke le cookie en l'associant au nom de domaine du site visité. Lorsque, le navigateur effectue une nouvelle connexion, possiblement longtemps après, il renvoie le cookie au serveur dans sa requête, si la date d'expiration n'est pas dépassée.

```
GET /page.php HTTP/1.1
Host: www.site.fr
Cookie: test=42;
```

Ainsi, le serveur peut enregistrer des données différentes sur chaque client.

Ces données sont ré-envoyées au serveur qui les a déposées. Cela permet au serveur de savoir qu'il a déjà vu ce client, et que la dernière fois que le client s'est connecté, le serveur lui a envoyé la valeur 42 pour le cookie test. Si la date d'expiration d'un cookie est passée, le navigateur Web le supprime. Si un cookie n'a pas de date, alors il est automatiquement supprimé lorsque le navigateur est fermé.

Ce mécanisme simple permet déjà de stocker des informations de personnalisation. Par exemple, lorsque l'on visite un site traduit en plusieurs langues, le serveur peut stocker dans un cookie la dernière langue utilisée par un client, et afficher automatiquement le site dans cette langue lors d'une visite suivante.

III.1.3 Session HTTP

Dans notre exemple initial, Alice a navigué sur la page d'accueil (une requête HTTP suivie d'une réponse), puis à navigué sur les pages restreintes (au moins une paire requête-réponse par page) puis a cliqué sur un bouton de déconnexion et s'est retrouvée sur la page d'accueil (nouvelle requête et réponse). Un tel ensemble de requêtes et réponses HTTP forment ce que l'on appelle une session HTTP. Une session commence par une phase de création de session, puis un ensemble de requêtes et réponses, puis se termine par une phase de destruction de session. L'intérêt est qu'une fois la session créée, le serveur se souvient des requêtes de la session, et ce pour chaque client. Dans notre exemple cela signifie que le serveur se souvient à chaque requête HTTP qu'Alice et Bob sont authentifiés, sans qu'ils aient à rentrer leur mot de passe à chaque fois (ce qui rendrait la navigation pénible).

D'un point de vue technique, les sessions sont implémentées grâce aux cookies. Lorsqu'un client se connecte à un serveur, ce dernier regarde si le client lui fournit un cookie particulier (appelons le ID). Si ce n'est pas le cas, il crée une nouvelle valeur unique appelée identifiant de session (par exemple 12345). Le serveur fait alors deux choses. Il garde en mémoire dans un dictionnaire global une entrée associant l'identifiant de session (ici 12345) à un dictionnaire de valeurs. Le serveur renvoie aussi le cookie ID=12345 au client. Lorsque le client se reconnecte, il renvoie son identifiant de session. Le serveur peut alors récupérer le dictionnaire de valeurs associé à l'identifiant et le passer au programme qui va générer la page Web. C'est de cette manière que l'on peut enregistrer pour chaque client des données qui restent côté serveur. Lorsque qu'un client se déconnecte, le serveur retire juste l'entrée associée à l'identifiant et efface les valeurs conservées.

Cela explique pourquoi les sites contenant un formulaire d'authentification ne fonctionnent pas lorsque les cookies sont désactivés dans les préférences du navigateur Web. Si le navigateur refuse de stocker les cookies, alors le serveur n'a plus moyen d'obtenir un identifiant unique pour ce navigateur de façon fiable. Attention, le stockage des cookies est effectué par le navigateur Web. Cela signifie que si on utilise le navigateur Firefox pour s'authentifier sur un site et que l'on exécute en parallèle un autre navigateur pour naviguer vers le même site, on ne sera pas authentifié dans le second navigateur.

Enfin, lorsque l'on utilise le mode de navigation privée du navigateur, ce dernier supprime, lors de la fermeture de la fenêtre, tous les cookies qui ont été créés depuis le démarrage de la navigation privée, même s'ils n'ont pas expiré.

III.1.4 Cookies et vie privée

Les cookies ont été proposés comme un moyen technique au dessus duquel implémenter d'autres mécanismes tels que les sessions. L'une des règles de sécurité initiale, toujours en vigueur, est que les navigateurs peuvent envoyer un cookie uniquement au domaine qui l'a déposé initialement. Cette règle visait à interdire à des sites tiers « d'espionner » les utilisateurs en sachant quels sites sont visités et

quelles valeurs sont dans les cookies (les dates en particulier peuvent donner un idée sur la fréquentation de tel ou tel site). Cependant cette règle peut être facilement contournée. Un site d'in- formation, par exemple www.news.com diffuse des articles variés. En dessous de chaque article, il propose des boutons « partager cet article sur reseau-social.com ou micro-blogging.net » (ou tout autre site). Les boutons contiennent des images, du code HTML et globalement des ressources accessibles à l'URL fournies par le site, comme `https://www.reseau-social.com/bouton?origin=123`. Lorsqu'un utilisateur navigue sur www.news.com et clique sur l'un de ses boutons, son navigateur effectue une requête HTTP vers `https://wuw.reseau-social.com/bouton?origin=123`. Ce site est donc dans la situation suivante

- il peut légitimement déposer un cookie le concernant, car l'utilisateur fait une requête vers son domaine ;
- il peut savoir quelle URL a été utilisée pour la requête, car c'est lui qui a fourni le code HTML à www.news.com. Il peut aisément connaître le site que l'utilisateur visitait voire le contenu du site au moment où l'utilisateur a partagé l'information. Il peut stocker toutes ces informations dans un cookie.

Lorsque l'utilisateur visite `https://www.reseau-social.com`, ce dernier peut légitimement relire son cookie et afficher, par exemple, de la publicité en rapport avec l'article lu. Les navigateurs permettent de limiter cette pratique via l'option « interdire les cookies de tierce partie ». Le navigateur refusera les cookies venant d'un domaine qui n'est pas celui affiché dans la barre d'adresse de l'URL.

Depuis mai 2018, le RGPD (règlement général sur la protection des données) encadre, entre autres, l'utilisation des cookies. Un utilisateur doit donner son consentement explicite et positif sur l'utilisation qu'un site fait des cookies, par catégories. Ainsi un utilisateur peut décider de n'accepter que les cookies « techniques » (ceux nécessaire à la mise en place d'une session) et refuser les autres cookies (publicité ciblée, télémétrie,...).