Introduction

Ce document explique le schéma http.pdf qui montre la totalité des échanges qui ont lieu lors de l'exécution de quelques pages web. Ce schéma résume l'usage des principes suivants :

- une page d'accueil en html statique
- du style css
- de l'exécution dans le navigateur : javascript
- de l'exécution sur le serveur (en php)
- un webservice
- un appel ajax
- des lectures dans une base de données

Quelque soit le langage, le framework ou les modes, ce qui est expliqué ici sur les échanges reste valable. Aussi il est utile de connaître ces principe car cela permet de résoudre de nombreux soucis des applications web.

Scénario

On demande la page d'accueil. Cette page contient uniquement un lien vers une autre page. Quand on le clique, on obtient la liste des prénoms des inscrits du site, mise en forme par un style css . On peut alors cliquer sur les prénoms pour afficher l'âge. Mais pour plus de dynamisme, l'affichage de l'âge est réalisé en ajax, technologie de moins de 10 ans qui ne rafraîchit pas toute la page mais seulement une zone d'intérêt. Ici cet ajax appelle un web-service qui lit en base de donnée l'âge de l'inscrit. Une popup apparaît alors pour afficher l'âge sans tout rafraîchir, ce qui est bien plus agréable.

Le schéma présenté montre les échanges sur un poste de développeur en local, mais une fois publié sur le web c'est identique.

0. Dans un navigateur on tape l'url du site. Comme on est en train de développer en local, on entre http://localhost, ce qui est par convention la machine locale. Le navigateur fabrique alors une requête (une demande), dont la forme est convenue et l'envoie au serveur web (ici, l'apache qu'on a installé sur le poste) cette requête est de la forme :
   ```
   GET / HTTP/1.1
   ```
   ce qui signifie que l'on souhaite récupérer (GET) la page d'accueil (/) puisqu'on a seulement entré le nom de domaine, et que l'échange réseau doit se faire en http.

1. Apache reçoit la requête et l'analyse. Ici comme aucun fichier précis n'est précisé et qu'on n'a indiqué que le nom de domaine, c'est une page d'accueil qui doit être servie. Par convention Apache va regarder dans ses fichier s'il y a un fichier index.html. Si oui, c'est cela qu'il va renvoyer dans sa réponse au navigateur, sinon un message du style « it works ». Dans le schéma on voit que ce fichier existe.

2. une réponse http est alors renvoyé au navigateur, contenant le contenu du fichier index.html précédé d'en-têtes qui précisent le format utilisé, la date de création du fichier, sa date de péremption, etc. La forme est :

```
HTTP/1.1 200 OK
Date: Thu, 26 Mar 2015 20:51:42 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Server: Apache
x-powered-by: PHP/5.4.38
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8

<html>
<body>
<a href=list.php>Liste</a>
</body>
</html>
```

3. Le navigateur recevant ce texte sait donc qu'il s'agit un fichier html grace à l'entête Content-Type. Il aurait aussi pu s'agir d'un contenu image, vidéo, audio, etc. C'est sa spécialité d' afficher le format html. Une page web décrite en html débute toujours par une balise (un marqueur) <html> et finit toujours par </html>, ce qui signifie « fin de html ». Le corps de la page (ce qui sera visible dans le navigateur) est décrit entre les balises <body> et </body>. Ici, il s'agit d'un lien web qui doit apparaître sous la forme du mot « Liste » et mener vers une autre page, list.php.

4. L'utilisateur clique sur le lien

5. Le navigateur envoie alors au navigateur une requête
   `GET /list.php HTTP/1.1`
   ce qui signifie qu'il veut avoir le contenu de cette page en retour.

6. Cette fois Apache ne va pas envoyer le contenu du fichier mais il va l'exécuter car il a repéré que le nom demandé se termine par .php. C'est donc une page html qui va être fabriquée à la volée. C'est ce qu'on appelle du contenu dynamique. L'exécution se passe sur le serveur, pas dans le navigateur. Ce programme exécuté a pour objectif de fabriquer du html, mais va y insérer des données, ici la liste des prénoms des inscrits du site. Voici le programme :

```
echo "<html>";
echo "<link href=monstyle.css>";
echo "<script src=monscript.js>";
echo "<body>";
$data=readDB_names();
foreach $ligne in $data
  {
  echo "<div>$ligne</div>";
  }
echo "</body>";
echo "</html>";
```

   On voit le mot echo souvent, qui signifie envoyer au navigateur ce qui suit. Le premier echo commence donc la fabrication du html.
   Le deuxième précise que le style visuel que devra avoir la page est décrit dans le fichier monstyle.css.
   Le troisième indique que le navigateur devra exécuter ce qui est décrit dans le fichier monscript.js. Le navigateur peut donc executer des choses, comme le serveur apache. Mais ces actions dans le navigateur concernent plutôt l'affichage et l'interaction avec ce que clique l'utilisateur, alors que le serveur calcule plutôt le contenu.
   Ensuite on démarre le corps de la page avec un echo « <body> » ;
   Ensuite le cœur du calcul de la liste à afficher, décrit après
   Enfin on finit la page avec /body et /html

On pourrait préciser des en-têtes http avec l'ordre header avant le premier echo, mais apache met déjà les principaux par défaut.

7. Le calcul commence par une lecture en base de données, résumé ci par la mention readDB_names() qui produit une requête à la base de données mysql, ici de la forme « select names from users », ce qui veut dire en langage standard SQL : donne la liste des noms enregistrés dans la table « users »

8. la base répond en donnant la liste demandée, à savoir : « kevin, brian, jason ». Cette information est stockée dans la variable php nommée $data. Ensuite par la commande foreach, on demande de faire ce qui est entre accolades pour chaque prénom trouvé. Ici, on fait un echo du prénom dans une zone carrée (div).

9. Entêtes mis à part, la réponse au navigateur est donc:

```
<html>
<link href=monstyle.css>
<script src=monscript.js>
<body>
<div>Kevin</div>
<div>Brian</div>
<div>Jason</div>
</body>
</html>
```

10. Le navigateur affiche donc cette page : trois zones carrée (div) contenant chacune un prénom.

11. Le navigateur sait qu'il doit donner à la page un certain style décrit dans le fichier monstyle.css (qu'il n'a pas encore). Il émet donc la requête

```
GET /monstyle.css HTTP/1.1
```

12. **Sans attendre la réponse (pour gagner du temps) le navigateur envoie aussi en parallèle une demande pour avoir le fichier monscript.js qu'on lui demande d'exécuter :**

```
GET /monscript.js HTTP/1.1
```

13. apache trouve le fichier monstyle.css

14. apache trouve le fichier monscript.js

15. apache renvoie le contenu du fichier de style au navigateur

16. apache renvoie le contenu du fichier de script au navigateur

17. Le navigateur applique le style, qui spécifie un bord de 1 pixel d'épaisseur et une police verdana gras. Il exécute le script trouvé dans le fichier monscript.js. Ce script ne fait rien tout de suite, mais prévient le navigateur que si l'on clique (OnClick) un prénom il faudra exécuter les choses suivantes, donc la syntaxe a été simplifiée ici :

```
OnClick :
var nom = div_content(clicked);
var age=CallWebService(nom);
popup(nom+" a "+age+" ans");
```

ce qui signifie :
extraire le contenu de la div cliquée (le prénom, donc) et le mettre dans la variable nom

appeler un webservice en lui précisant le nom comme entrant, et mettre le resultat reçu dans la variable age
afficher une popup dont le texte sera « untel a X ans ».
Ceci n'est pas exécuté avant que l'on clique un prénom.

18. On clique un prénom, Jason, ce qui déclenche l'**ajax**.

19. Le javascript est donc déclenché, il extrait le texte « Jason » et envoie au serveur une requête qui contient un paramètre :
```
GET /webservice.php?name=Jason HTTP/1.1
```

20. apache voit que c'est un php, à exécuter donc. Il trouve le fichier demandé, webservice.php et exécute les trois commandes qui y sont :
```
$nom=getParam("name")
$age=readDB_age($nom);
echo encode_json($age);
```
La première demande d'extraire le paramètre « name » de la requête http GET reçue et de stocker la valeur dans la variable php $nom.
NB : En réalité cette ligne s'écrit en php : $nom = `$_GET['name']`

21. une requête sql est fabriquée par la commande `readDB_age($nom)` de la forme :
```
select age from users where name='jason'
```

22. la base répond 22, qui est l'âge de l'utilisateur Jason. Cette valeur est stockée dans la variable php $age.
Le php fait ensuite un echo pour donner sa réponse, mais n'a pas besoin de mettre des balises html car le webservice est destiné à être utilisé par des machines ou des programmes, pas à être joli dans un navigateur. Ce chiffre n'est quand même pas renvoyé tel quel mais est enrobé un peu, c'est ce qu'on appelle le format json. Ce format de transport de données est à la mode pour les réponses de webservices.

23. Un json contenant la valeur 22 est donc renvoyé au navigateur en guise de réponse :
```
HTTP/1.1 200 OK
Server: Apache
x-powered-by: PHP/5.4.38
Content-Type: text/json; charset=UTF-8

{"age":"22"}
```

24. Le json est décortiqué et la valeur 22 est stockée dans la variable javascript age. Le navigateur exécute la troisième ligne du script : afficher la popup avec le résultat.

Conclusion :
La syntaxe est volontairement minimaliste mais exacte.

Librairies : Ici on a utilisé des fonctions toutes faites comme CallWebService ou readDB. On peut trouver comment les créer sur de nombreux forums, par exemple :
http://notos.fr/blog/index.php?article23/exemple-de-web-services-en-php
http://www.tizag.com/mysqlTutorial/mysqlquery.php

Il est vraiment utile de comprendre la mécanique décrite ici, car ça permet de gagner énormément de temps pour le développement. Et 90% reste valable quelque soit les technologies utilisées. On

pourra voir tous les échanges http dans le navigateur (safari, firefox, chrome...) en faisant un bouton droit / inspecter l'élément et en choisissant l'onglet réseau ou activité. Ainsi quand on a un bug on peut vite comprendre où, car on sait ce qui devrait normalement passer sur le réseau.

Pour le développement, l'utilisation d'un framework réduit un peu la quantité de code à écrire mais sont intérêt est surtout qu'il contraint la manière de programmer, ce qui est utile pour le développement en équipe. Aussi le développeur se concentre sur la fonctionnalité. Le désavantage est qu'il faut du temps pour comprendre la philosophie du framework. Celui qui est à la mode en ce moment pour le monde php est symfony. Son but est de séparer le code de présentation (fabrication du html) du code « métier » (fonctionnalités et règles liées au business + accès aux données). Il facilite la partie base de donnée.

Installation d'un poste de développement : pour développer il est plus facile d'installer sur son poste les mêmes logiciels que ceux qui font fonctionner les vrais sites web chez les hébergeurs. On crée alors un site web local, et quand il est au point on le publie en copiant les fichiers créés sur un vrai serveur web chez l'hébergeur. Il faut ici installer donc la trilogie classique apache/mysql/php, très répandue sur le web. Par facilité on installe un pack comme AMPPS, Wamp ou EasyPHP qui est déjà complet et pré-configuré.

Anselme Dewavrin