NSI 1ère - IHM sur le web

QK

Le WEB

Le “World Wide Web”, plus communément appelé “Web” a été développé au CERN (Conseil Européen pour la Recherche Nucléaire) par le Britannique Sir Timothy John Berners-Lee et le Belge Robert Cailliau au début des années 90.

Tim Berners-Lee met au point le système hypertexte qui permet, à partir d'un document, de consulter d'autres documents en cliquant sur des mots clés (liens).

Tim Berners-Lee développe le premier navigateur web (logiciel permettant de lire des pages contenant des hypertextes).

Techniquement le web se base sur trois choses : le protocole HTTP (HyperText Transfert Protocol), les URL (Uniform Resource Locator) et le langage de description HTML (HyperText Markup Language).

Web != internet

- **internet** : réseau mondial de machines inter connectées.
- **web** : http + url + html. Les pages qu'on peut visiter dans le navigateur.

Par exemple, envoyer un email depuis sa machine n'utilise pas le web mais un autre protocole (smtp).

URL

Une URL (Uniform Ressource Locator) permet d'identifier une ressource sur un réseau.

Une URL indique “l'adresse” où se trouve la ressource (un fichier, un dossier etc.)

La structure qu'on emploie est en arboresence (fichier dans un dossier dans un dossier etc.)

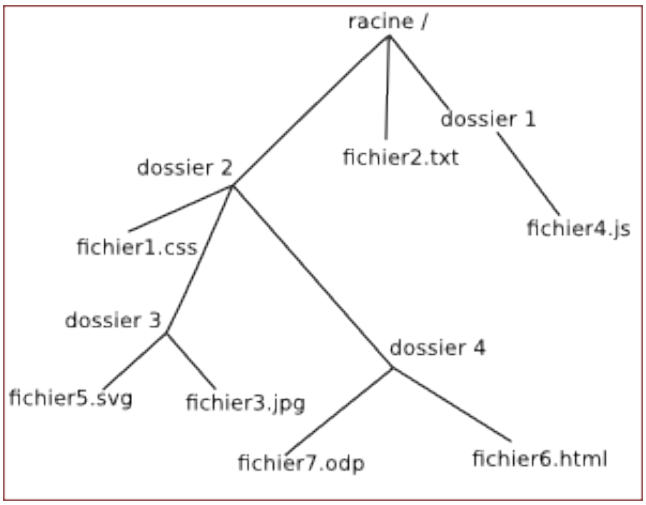

Figure 1: arborescence

La base de l'arbre est la racine, généralement notée `/`

Chemin absolu, chemin relatif

- Chemin absolu : depuis la racine. Exemple : `/dossier2/dossier3/fichier3.jpg`
- Chemin relatif : depuis l'endroit où on se trouve. Exemple, depuis `fichier1.css` : `dossier3/fichier3.jpg`

HTML et CSS

- HTML : format de données reçues pour représenter une page web. Langage de balises permettant d'écrire de l'HyperText. Permet de structurer une page et d'inclure des ressources.
- CSS : langage permettant de mettre en forme une page web. Utilise aussi des balises mais dans une syntaxe différente de l'html.

HTML n'est pas un langage de programmation (pas de boucle, de variables etc.) mais de description.

Exemple

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>nom de la fenêtre</title>
    <link rel="stylesheet" href="style.css">
    <script src="monage.js" charset="utf-8"></script>
  </head>
  <body>
    <h1>Un titre</h1>
    <h2 class=rouge>Un sous titre</h2>
    <ul>
      <li>premier élément d'une liste non ordonnée</li>
      <li>second élément</li>
    </ul>
    <p>
      paragraphe contenant
      <a href="https://qkzk.xyz">un lien</a>
      vers mon site.
    </p>
    <div class="rouge">
      <p id=agerobert>mon age</p>
    </div>
  </body>
</html>
```

et le css qui l'accompagne

```
h1 {
  color: green;
  text-align: center;
}

.rouge {
  color: white;
  background-color: red;
}

#agerobert {
  font-weight: bold;
}
```

- Le titre (dans la page) est en couleur verte.
- Le sous-titre et la dernière div sont en blanc sur fond rouge.
- Le texte "mon age" est en gras.

On peut construire l'arborescence d'une page web :

```
html
  |_ head
  |    |_ meta
  |    |_ title
  |    |_ link
  |    |_ script
  |_ body
```

```
|_ h1
|_ h2
|_ ul
|  |_ li
|  |_ li
|_ p
|  |_ a
|_ div
   |_ p
```

JavaScript

Présentation

HTML et CSS sont accompagnés de JavaScript.

JavaScript (aucun rapport avec Java) est :

- le seul langage que les navigateurs (chrome, firefox etc.) puissent exécuter.
- un langage de script (comme Python) dont la syntaxe dérive du C (des `{}` partout)

JavaScript remonte aux année 90 et a longtemps été perçu comme un accessoire.

Il est maintenant le langage préféré des développeurs et le langage le plus utilisé dans le monde.

JavaScript permet de faire faire les gros calculs (d'un jeu vidéo, d'une animation) sur la machine **DU CLIENT** et économise beaucoup de ressources pour le serveur.

Sans JavaScript, pas de réseaux sociaux, pas d'internet moderne, aucun serveur ne pourrait faire autant de calculs.

JavaScript côté client

Côté client, JavaScript est exécuté par le navigateur. Il est très limité. Il ne peut pas accéder aux fichiers de la machine (sinon imaginez le désastre : vous accédez à un site qui efface vos fichiers sans vous prévenir !).

On charge un fichier JS depuis une page web avec la balise `<script>`

Quelques éléments. Le fichier `monage.js`

```
var nom = "Robert"; // var : on crée une variable. ; fin d'instruction. // commentaire.
var anneeNaissance = 1914;
for (var k = 0; k < 3 ; k++){
  console.log("bonjour !"); // une boucle de k = 0 à k = 2. On écrit dans la console, rien dans la page
}
function calculAge(anneeNaissance, anneeCourante){
  // une fonction qui renvoie le résultat d'un calcul
  return anneeCourante - anneeNaissance;
}
document.getElementByID('agerobert').innerHTML = calculAge(1515, 2019);
// on cherche dans le document...
// l'élément dont l'id est 'agerobert'
// on change son contenu html (.innerHTML)...
// à la place on met la valeur de retour de la fonction
// il est sacrément vieux Robert...
```

JavaScript côté serveur

Node.js est un serveur web très populaire tournant en JavaScript. Il dispose de nombreux avantages sur ses concurrents plus anciens : montée en charge (passer de 10 utilisateurs à 10.000.000), gestion d'événements (quand la vidéo est uploadée, arrête la pub et envoie un message au client etc.)

JavaScript sur le bureau

De nombreuses applications utilisent Electron (moteur de fenêtre écrit en javascript) pour créer et animer des fenêtres. Citons Atom (sublime text en mieux) d'où j'écris ces lignes.

Modèle client-serveur

Dans un réseau, on distingue généralement le serveur (qui fournit une ressource) du client (qui consomme la ressource).

- vous ouvrez le navigateur... et tapez `http://qkzk.xyz`
- votre machine se connecte au serveur...
- qui envoie une page web `index.html`...
- votre navigateur (le client) la décode et l'affiche.

Ces échanges sont formatés et se déroulent comme un dialogue.

Chaque étape d'une connexion réseau se déroule généralement ainsi.

Modèle P2P

Certains réseaux dits "de pair à pair" mettent toutes les machines à égalité.

Le réseau n'est plus centré autour d'un serveur mais est un maillage de machines qui sont toutes serveur et client.

Citons par exemple : les DNS (qui traduisent qkzk.xyz en 185.199.110.15), le réseau local de la Nintendo 3DS, les cryptomonnaies (bitcoin, ethereum etc.) et bien sûr bittorent...

Serveur

Un site web important aura besoin de plusieurs machines qui exécutent toute le même programme. Cela lui permet de traiter des dizaines de milliers de connexion par seconde.

Les serveurs sont généralement spécialisés. Ils ne font qu'une seule chose, le plus efficacement possible.

Protocole HTTP

Le web utilise le protocole réseau HTTP pour échanger des informations.

C'est un protocole "haut niveau", cela signifie qu'il n'a pas besoin de savoir si vous êtes connecté en ethernet (câble RJ45), en wifi ou en 4G pour échanger des informations. Google affiche la même page que vous ayez activé le wifi ou non !

Les échanges HTTP se font par des requêtes du client auxquelles le serveur répond.

Requête : client ——> serveur

Une requête, c'est DU TEXTE (encodé en binaire of course) qui contient :

- la méthode employée pour effectuer la requête
- l'URL de la ressource
- la version du protocole utilisé par le client (souvent HTTP 1.1)
- le navigateur employé (Firefox, Chrome) et sa version
- le type du document demandé (par exemple HTML)
- ...

Voici un exemple de requête HTTP :

```
GET /mondossier/monFichier.html HTTP/1.1
User-Agent : Mozilla/5.0
Accept : text/html
```

la méthode est ici GET.

Méthodes

Il en existe au moins 9

GET, HEAD, POST, OPTIONS, CONNECT, TRACE, PUT, PATCH, DELETE

Mais on en rencontre surtout 4 :

- GET : C'est la méthode la plus courante pour demander une ressource. Elle est sans effet sur la ressource.
- POST : Cette méthode est utilisée pour soumettre des données en vue d'un traitement (côté serveur). Typiquement c'est la méthode employée lorsque l'on envoie au serveur les données issues d'un formulaire.
- DELETE : Cette méthode permet de supprimer une ressource sur le serveur.
- PUT : Cette méthode permet de modifier une ressource sur le serveur

en fait >90% du temps c'est GET ensuite POST et parfois les autres.

Réponse : serveur ——> client

Une fois la requête reçue, le serveur va renvoyer une réponse. Voici un exemple de réponse du serveur :

```
HTTP/1.1 200 OK
Date: Thu, 15 feb 2019 12:02:32 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
<!doctype html>
<html lang="fr">
<head>
<meta charset="utf-8">
<title>Voici mon site</title>
</head>
<body>
 <h1>Hello World! Ceci est un titre</h1>
<p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
</body>
</html>
```

La description de la réponse est ce qui précède `<!doctype html>`

- le serveur renvoie du code HTML, encodé en utf-8
- HTTP/1.1 formaté en HTTP/1.1
- 200 OK : le code réponse signifiant que tout va bien

  Les autres codes courant sont `404` : ressource introuvable, `403` : ressource interdite, `50x` : erreur du serveur.

Créer un serveur web en Python avec Flask

Tous les langages modernes permettent de créer des serveurs web. Servir une page = la rendre accessible à d'autres machines de votre réseau.

Python propose plusieurs serveurs :

- un natif (sans librairie extérieure) mais pauvre,
- Flask : léger et modulaire (on installe des librairies supplémentaires pour les opérations spécifiques),
- Django : lourd et puissant (tout est déjà dedans).

On utilisera **Flask**. Vous l'utiliserez dans vos projets.

Exemple

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
  return "<p>Tout fonctionne parfaitement</p>"

app.run(debug=True)
```

Lancez le script et vous avez une page web à l'adresse `http://localhost:5000`

Détaillons :

`from flask import Flask`

Nous importons la bibliothèque Flask

`app = Flask(__name__)`

Nous créons un objet app : cette ligne est systématique nécessaire.

```
@app.route('/')
```

Décorateur. La fonction qui suit sera exécutée si un client se connecte à la racine du site.

```
def index():
  return "<p>Tout fonctionne parfaitement</p>"
```

Cette page renvoie du texte, le contenu de la page.

Exemple plus élaboré. Méthode GET pour remplir un formulaire et l'afficher.

3 fichiers sont nécessaires :

index.html

```
<!doctype html>
<html lang="fr">
   <head>
       <meta charset="utf-8">
       <title>Le formulaire</title>
   </head>
   <body>
       <form action="http://localhost:5000/resultat" method="get">
               <label>Nom</label> : <input type="text" name="nom" />
               <label>Prénom</label> : <input type="text" name="prenom" />
               <input type="submit" value="Envoyer" />
       </form>
   </body>
</html>
```

resultat.html

```
<!doctype html>
<html lang="fr">
   <head>
       <meta charset="utf-8">
       <title>Résultat</title>
   </head>
   <body>
       <p>Bonjour {{prenom}} {{nom}}, j'espère que vous allez bien.</p>
   </body>
</html>
```

views.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
 return render_template('index.html')

@app.route('/resultat',methods = ['GET'])
def resultat():
  result=request.args
  n = result['nom']
  p = result['prenom']
  return render_template("resultat.html", nom=n, prenom=p)

app.run(debug=True)
```

Attention à

1. `render_template` dans le script Python. Il sert un fichier HTML en remplaçant du contenu par les variables `nom=n` et `prenom=p`
2. Aux `{{ }}` dans `resultat.py`. `{{ prenom }}` sera remplacé par le contenu de la variable prénom.
3. Au trajet de l'information : formulaire (GET) -> serveur (`result.args`) -> réponse