

MobApp – HTML, CSS, Javascript

Laurent Daverio, Olivier Hermant



PSL 

Centre de recherche
en informatique

Plan

1. XML

2. HTML

3. CSS

4. JS

1. XML

XML - eXtensible Markup Language

Classification des espèces :

```
<règne>
  <animal>
    ...
    <mammifère>
      <homoSapiens bras="2" jambes="2" ailes="0">
        <formation>
          <ingénieur ecole="mines">
            ...
          </ingénieur>
          <insp />
        </formation>
      </homoSapiens>
    </mammifère>
  </animal>
  <champignons>
    ...
  </champignons>
</règne>
```

XML – composition chimique

- ▶ forme arborescente
- ▶ tag/balise : `<animal>`
- ▶ balises ouvrantes (`<animal>`) **doivent** se fermer (`</animal>`)
 - ▶ cas particulier : balises autofermantes “stériles” (`<insp />`)
- ▶ les balises peuvent avoir
 - ▶ des attributs :
`<homoSapiens bras="2" jambes="2" ailes="0">`
 - ▶ des descendants (si non autofermantes)

Organiser les informations.

2. HTML

HTML : HyperText Markup Language

- ▶ inventé en Europe (CERN)
- ▶ ancêtre de XML : mêmes principes
- ▶ **pages web** : avec des [liens hypertexte](#)
- ▶ DÉMO: l'inspecteur de code

Une page HTML assez minimale

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8" />
  <title>Page Vide</title>
</head>
<body>
  <!-- Ceci est un commentaire dans une page vide -->
</body>
</html>
```

- ▶ séparation entre en-tête `<head></head>` (non affichée) et corps `<body></body>`
- ▶ `[page_vide.html]`
- ▶ que l'on peut ouvrir dans un navigateur

Quelques balises

- ▶ organisation du contenu en *divisions*

```
<div>contenu</div>
```

- ▶ et en *paragraphes*

```
<p></p>
```

- ▶ *passer à la ligne* avec

```
<br />
```

- ▶ les *liens*

```
<a href="URL">contenu</a>
```

- ▶ [page_div.html]

D'autres balises

- ▶ liste non ordonnée (`...`) ou ordonnée (`...`)
 - ▶ chaque item de la liste (`...`)
- ▶ un tableau à 5 lignes et 2 colonnes:

```
<table>
<tr> <td> (1,1) </td> <td> (1,2) </td> </tr>
<tr> <td> (2,1) </td> <td> (2,2) </td> </tr>
<tr> <td> (3,1) </td> <td> (3,2) </td> </tr>
<tr> <td> (4,1) </td> <td> (4,2) </td> </tr>
<tr> <td> (5,1) </td> <td> (5,2) </td> </tr>
</table>
```

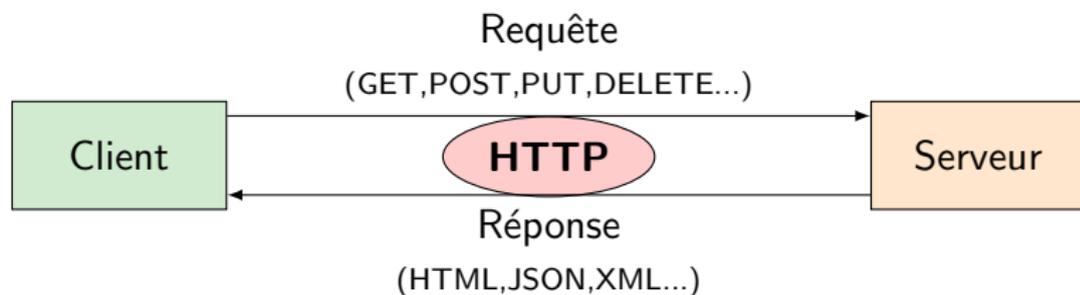
- ▶ [[page_liste.html](#)]
- ▶ les balises ne sont pas affichées. Caractères spéciaux :

< ~> <

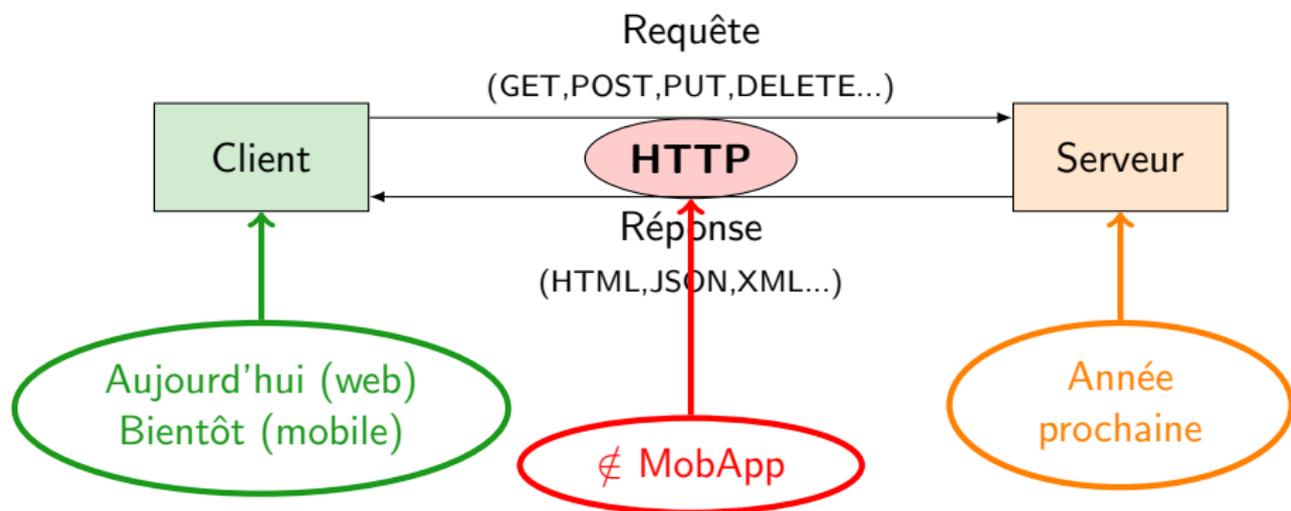
> ~> >

⋮ ~> ⋮

Le modèle client–serveur



Le modèle client–serveur



Les formulaires

- ▶ comment envoyer des requêtes au serveur ?
- ▶ la balise `<form>` et ses attributs
 - ▶ `method` : restreint à GET ou POST
 - ▶ `action` : l'URL où sera envoyée la requête
- ▶ la balise `<input>` et ses attributs
 - ▶ `type`
 - ▶ `name` : le nom du paramètre de la requête
 - ▶ `value` : sa valeur
- ▶ la balise `<button>` et ses attributs
 - ▶ `type` (submit pour envoyer le formulaire)
- ▶ bien sûr, il faut un serveur en face
- ▶ [[page_formulaire.html](#)]

3. CSS

CSS – Cascading Style Sheet

- ▶ l'attribut `style`
 - ▶ trop de propriétés pour les lister
 - ▶ schéma

`propriété : valeur;`

- ▶ [[page_style.html](#)]
- ▶ SOC : séparer information, organisation, et présentation
- ▶ factorisation dans `<head>`, dans une balise `<style>` où l'on référence les balises du corps de la page :
 - ▶ par leur identifiant (attribut `id`)
 - ▶ par leur type
 - ▶ par leur classe (attribut `class`)
- ▶ et bien d'autres choses...
- ▶ [[page_css.html](#)]

Fichiers séparés

Encore trop lourd

- ▶ mettre le CSS dans un fichier séparé

```
<link rel="stylesheet" href="style1.css">
```

- ▶ [page_css_separee.html]

Fichiers séparés

Encore trop lourd

- ▶ mettre le CSS dans un fichier séparé

```
<link rel="stylesheet" href="style1.css">
```

- ▶ [page_css_separee.html]

- ▶ on peut mettre du style à la fois dans

- ▶ un fichier séparé, le header (balise `<style>`), les attributs `<XX style="...">`
- ▶ priorités : *Cascading*

4. JS

Un bouton sans formulaire

- ▶ `[page_auteur.html]`

Un bouton sans formulaire

- ▶ [[page_auteur.html](#)]

JAVASCRIPT

- ▶ langage de programmation
- ▶ rien à voir avec Java (sauf le nom)
- ▶ exécuté (interprété) localement, **par le navigateur**

Un bouton sans formulaire

- ▶ [[page_auteur.html](#)]

JAVASCRIPT

- ▶ langage de programmation
- ▶ rien à voir avec Java (sauf le nom)
- ▶ exécuté (interprété) localement, **par le navigateur**
- ▶ accès aux éléments du document
 - ▶ DOM: Document Object Model
 - ▶ `let aut = document.querySelector('#auteur')`
 - ▶ sélectionne la *première* balise de type `<... id="auteur" ...>` rencontrée
 - ▶ `#` sélectionne par id. Par type, nom, classe possible (cf. syntaxe CSS)
 - ▶ balise manipulée en tant qu'objet
 - ▶ lire/modifier les attributs, les enfants...

```
aut.innerHTML='Olivier Hermant'  
aut.style.backgroundColor = 'red'  
aut.style.fontSize = '300\%'
```

Javascript rapide

- ▶ nous n'allons pas apprendre JS en 1h

Javascript rapide

- ▶ nous n'allons pas apprendre JS en 1h
- ▶ mais en 5 minutes :
 - ▶ déclarer une variable : `let`, sans type
 - ▶ ; obligatoire sauf si on change de ligne
 - ▶ 'une chaîne' et "une chaîne"

Javascript rapide

- ▶ nous n'allons pas apprendre JS en 1h
- ▶ mais en 5 minutes :
 - ▶ déclarer une variable : `let`, sans type
 - ▶ ; obligatoire sauf si on change de ligne
 - ▶ 'une chaîne' et "une chaîne"
- ▶ SOC : mettre le JS dans une balise `<script>`
[page_auteur_2.html]
- ▶ fichiers séparés encore mieux, à appeler dans l'en-tête
[page_auteur_3.html]

JS et le modèle client–serveur

- ▶ AJAX : Asynchronous JavaScript and XML
- ▶ en JS :
 - ▶ envoi de la requête [à moitié à la main]
 - ▶ traitement de la réponse (une fois reçue) [automatisé]
 - ▶ mise à jour de la page [à la main]
- ▶ utilisation de **bibliothèques** :
 - ▶ oui pour l'envoi des requêtes (**fetch**, **axios**)
 - ▶ non pour le reste (notamment UI)
 - ▶ par ordre chronologique : jQuery, Angular, Node
 - ▶ évolution vers des bibliothèques complètes
 - ▶ web : Vue, React
 - ▶ mobile : React Native, Ionic, Flutter
 - ▶ etc.

Attribution dynamique de propriétés/code : callbacks

- ▶ sélecteur du DOM `document.querySelector()`
- ▶ l'objet sélectionné a de nombreux attributs
- ▶ exemple : [[page_statique.html](#)]
 - ▶ lecture/écriture des attributs

Attribution dynamique de propriétés/code : callbacks

- ▶ sélecteur du DOM `document.querySelector()`
- ▶ l'objet sélectionné a de nombreux attributs
- ▶ exemple : [[page_statique.html](#)]
 - ▶ lecture/écriture des attributs
- ▶ faire disparaître `onClick` et l'appel en dur à `changer()`
 - ▶ [[page_dyn.html](#)]

Attribution dynamique de propriétés/code : callbacks

- ▶ sélecteur du DOM `document.querySelector()`
- ▶ l'objet sélectionné a de nombreux attributs
- ▶ exemple : [[page_statique.html](#)]
 - ▶ lecture/écriture des attributs
- ▶ faire disparaître `onClick` et l'appel en dur à `changer()`
 - ▶ [[page_dyn.html](#)]
- ▶ `addBehavior` affecte une *fonction* à l'attribut `onclick`
- ▶ Possible seulement si le document est **entièrement** chargé
 - ▶ il faut retarder l'appel à `addBehavior`
 - ▶ donc enregistrer `addBehavior` comme **fonction de callback**

Faire des requêtes en JS

- ▶ principe : clic $\xrightarrow{\text{js}}$ requête $\overset{\text{wait}}{\rightsquigarrow}$ réponse $\xrightarrow{\text{js}}$ callback $\xrightarrow{\text{js}}$ contenu
- ▶ fetch : standard de JS
- ▶ utilisation plus simple par **axios**
- ▶ format de retour : JSON (standard)
- ▶ **enregistrer une fonction de callback**
 - ▶ en charge de la mise à jour de la page
 - ▶ en fonction de la réponse reçue
 - ▶ ... quand elle sera reçue
- ▶ [[page_requete.html](#)]
- ▶ noter l'utilisation de *fonctions anonymes*

Outils de base pour développer avec HTML, CSS, JS

- ▶ un bon éditeur (e.g. vscode)
- ▶ inspecteur de code & outils développeur des navigateurs