Langages pour le web

DIU Enseigner l'Informatique au Lycée

C. Parent-Vigouroux, B. Wack

UFR IM2AG, Université Grenoble Alpes

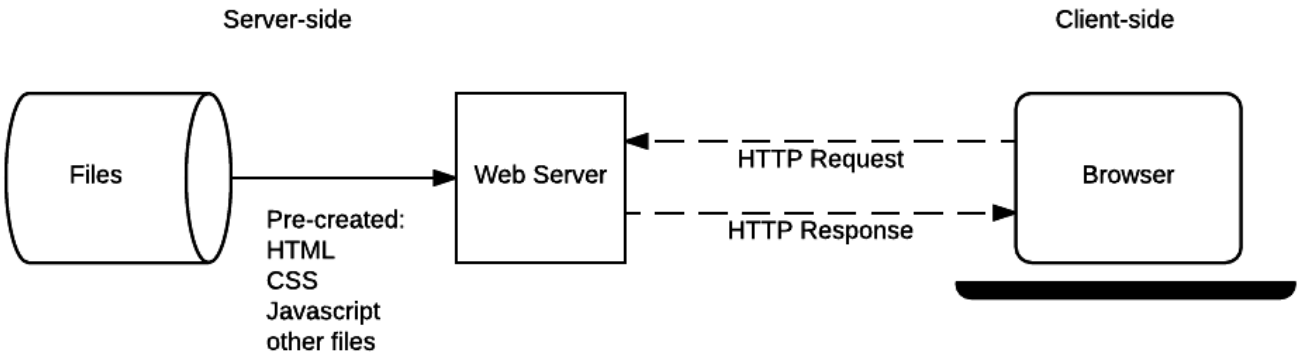
automne 2020

Plan

Plan

Version basique

1. Un client demande une page HTML par le biais d'une requête HTTP.
2. Le serveur cherche la page dans ses fichiers et l'envoie.



Source : `https://developer.mozilla.org/fr/docs`

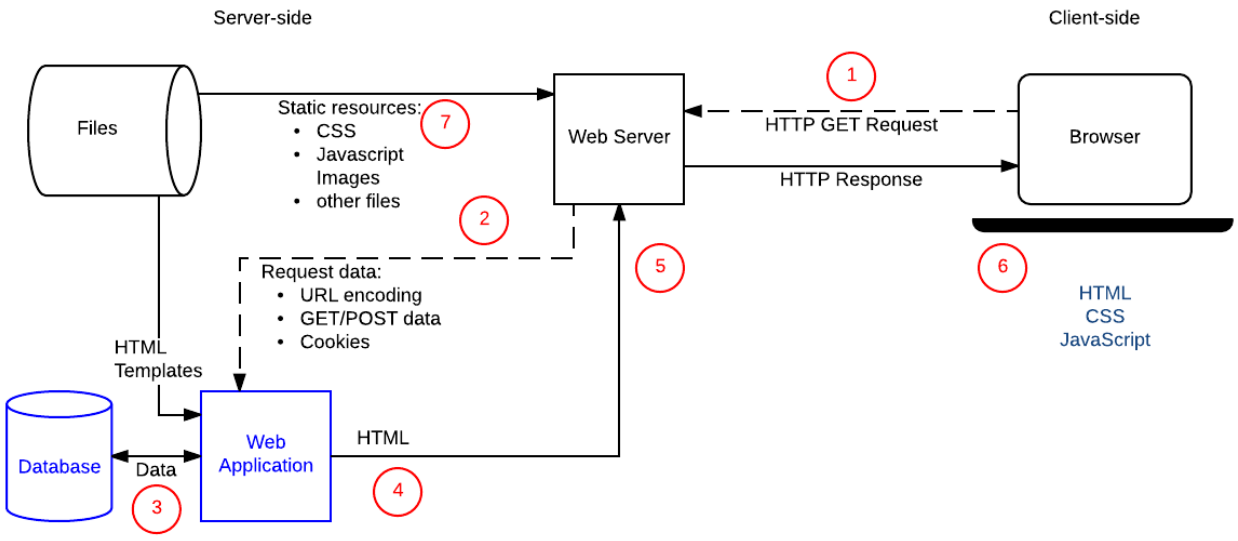
Limitations :

- la page est toujours la même
- beaucoup de travail pour le serveur
- temps d'attente parfois long avant d'avoir une réponse

Version adaptative côté serveur (hors sujet pour nous)

1. Le client spécifie également des *paramètres* pour la page demandée.
2. Le serveur peut
   - exécuter du code (par exemple du PHP)
   - consulter une base de données

   pour construire le HTML à envoyer.



Contenu beaucoup plus flexible mais :

- encore plus de travail pour le serveur !
- et donc de temps d'attente...

Côté client

Le serveur envoie en fait généralement 3 types de contenu :

- HTML (HyperText Markup Language) :
  structure et contenu du document
- CSS (Cascading Style Sheets) :
  apparence du document
- JavaScript :
  code exécutable par **le client**

Avantages

- Séparation de la forme et du fond
- Charge de travail répartie entre les clients
- Plus de temps de latence

mais le serveur reste le seul à pouvoir consulter directement la base de données !

Plan

HTML : éléments

- Un langage formel pour décrire des données, pas un algorithme
- Fait partie de la famille des langages à *balises* :
  - `<truc> ... </truc>` constitue un élément de la page
  - Structure arborescente : un élément peut contenir un ou plusieurs éléments, etc.

Quelques balises courantes :

- `<html>`, `<body>` et `<head>` : indispensables dans toute page HTML
- `<h1>`, `<h2>`... définissent des titres de différents niveaux
- `<p>`, `<div>`, `<span>`... permettent de structurer la page
- Certaines balises comme `<br>` n'ont pas besoin d'être fermées (on peut aussi écrire `<br />`)

HTML : les attributs

Une balise peut contenir des attributs qui en précisent le comportement :

```
<a href="https://pep8.org/">Une page bien utile</a>
```

toujours de la forme `cle="valeur"`

Deux attributs importants :

- `id="..."` associe un identifiant unique à un élément de la page
- `class="... ..."` affecte une ou plusieurs « catégories » aux éléments d'une page

Attention

De nombreux attributs (et balises) modifient l'apparence des éléments, ce qui est à éviter (HTML ne devrait concerner que le contenu, pas son apparence).
On pourra cependant utiliser des balises comme `<strong>` ou `<em>` qui décrivent « l'importance » attachée à un élément, sans préjuger de la façon dont on marquera cette importance.

Structure de la page et flot

On différencie deux types d'éléments :

- *block*
  - commencent « à la ligne »
  - prennent toute la largeur disponible
  - dimensions et position réglables

  `<div>` mais aussi `<h1>`, `<p>`...
- *inline*
  - s'insèrent dans le flot courant
  - réglages de dimensions sans effet

  `<span>` mais aussi `<a>`, `<em>`...

Un *block* ne peut pas être contenu dans un *inline*.

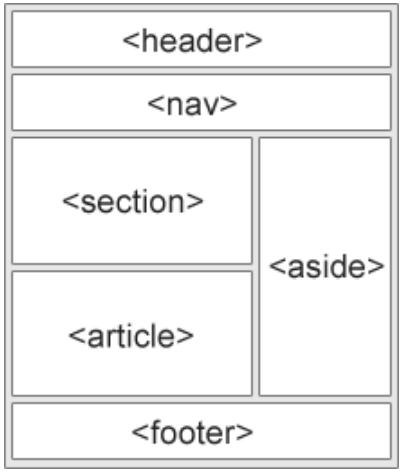
OK pour du texte « au kilomètre » mais peu adapté pour une composition plus complexe ; problème de l'affichage adaptatif (PC, tablettes, smartphone...)

Mise en page avancée (HTML5)

- nouveau type *flex* qui permet
  - de placer des éléments côte à côte sur une même ligne/colonne
  - d'adapter leur taille selon leur contenu et la place disponible
  - de définir leurs tailles respectives...

Raphaël Goetter, https://www.alsacreations.com

- nouveaux conteneurs pour les éléments les plus courants dans la structure d'une page



https://www.w3schools.com/html/html_layout.asp

Plan

Principes de CSS

- Encore un langage formel de description des *données* !
- Détermine l'apparence qu'auront certains éléments d'une page
- Focus sur la réutilisabilité :
  - permet de spécifier (et de modifier) l'apparence de toute une classe d'éléments
  - la même feuille de style peut servir pour plusieurs pages
  - inversement la même page peut faire appel à plusieurs feuilles de style

Éléments de syntaxe

Une feuille de style est constituée de règles de la forme :

```
sélecteur {
  prop1: valeur1 ;
  prop2: valeur2 ;
  ...
}
```

- le sélecteur détermine quels éléments seront concernés par la règle
- chaque propriété correspond à une marque de mise en forme :
  - couleur : `color`, `background-color`...
  - taille : `width`, `height`...
  - comportement de l'élément dans le flot : `display:` `block` ou `inline`
  - bordures, etc.

Les sélecteurs

- `ty` : tous les éléments du type `ty` (par exemple `div` ou `h2`)
- `#id` : l'élément d'identifiant `id`
- `.cl` : tous les éléments ayant la classe `cl`

Les sélecteurs peuvent également être combinés :

- `ty.cl` : les éléments de type `ty` ayant la classe `cl`
- `s1, s2, s3` : les éléments correspondant à au moins un de ces sélecteurs
- `s1 s2` : les éléments `s2` contenus dans un élément `s1`
- etc.

Plan

Outils de développement web

Dans Mozilla Firefox :
Menu Outils -> Développement Web OU appui sur F12

- Inspecteur de code HTML
  (aussi accessible par clic droit -> Examiner élément)
- Console et Débogueur Javascript
- Éditeur de style CSS
- Analyseurs de performances
- etc.

Plus d'infos sur :
`https://developer.mozilla.org/fr/docs/Tools`

Voir aussi `https://validator.w3.org/#validate_by_upload`

Références

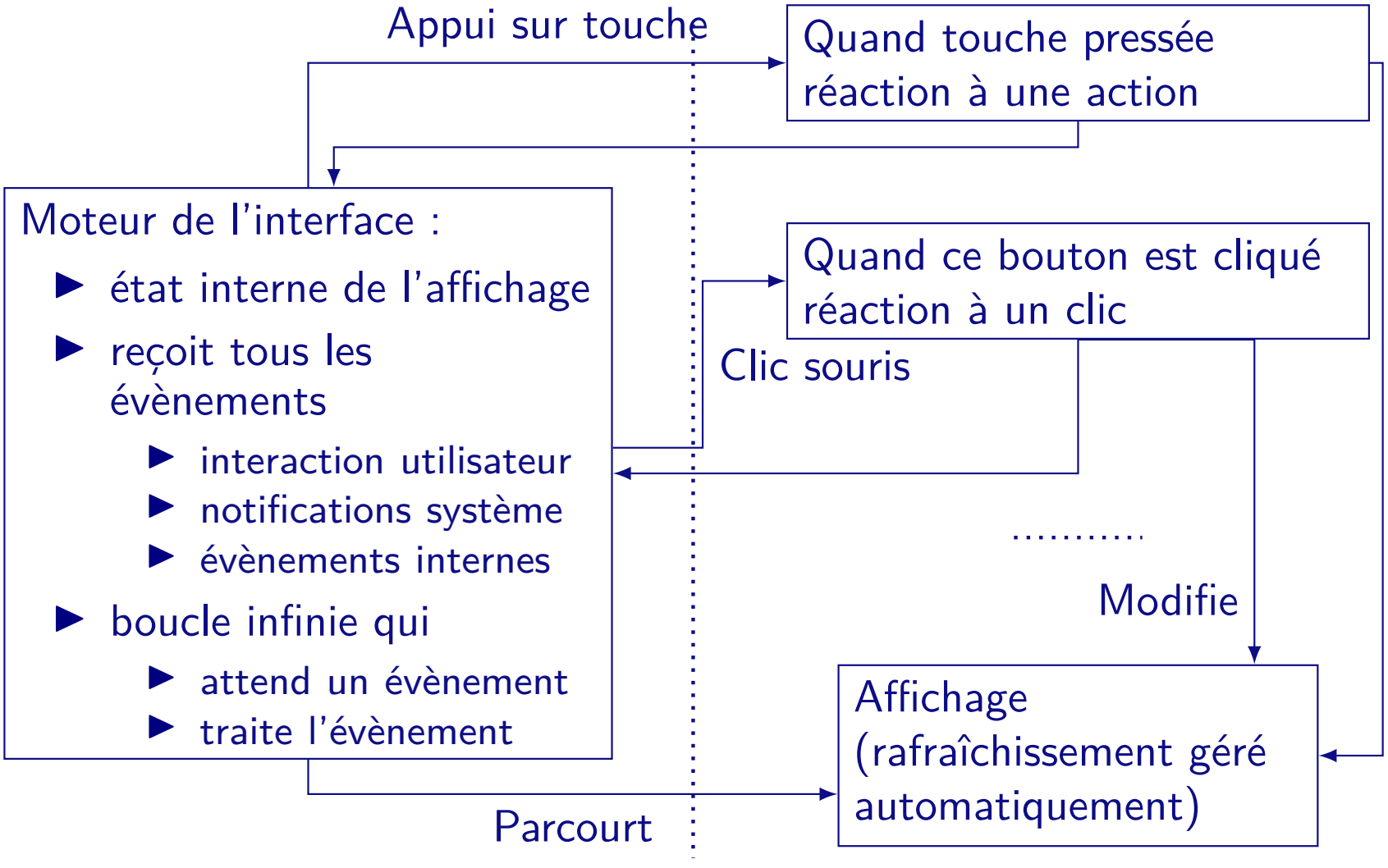
- Documentation Mozilla
  `https://developer.mozilla.org/fr/docs/Web`
  (attention, certaines pages ne sont pas traduites)
- Tutoriels et référence (en anglais mais beaucoup d'exemples) :
  `http://www.w3schools.com/`
- Apprendre JavaScript avec un livre en ligne :
  `https://javascript.info` ou `http://exploringjs.com`
- Un index unifié `https://devdocs.io/`

Quelques astuces

- Pour tester une mise en page, en particulier avec des blocs `flex` :
  - redimensionner la fenêtre du navigateur
  - ou (sous Firefox) taper `Ctrl+Shift+M`
- Pour positionner un élément :
  1. lui attribuer une classe `cl`
  2. définir une règle CSS
     `cl {background-color: red;}`
     pour le visualiser dans la page
- Si une règle CSS semble sans effet :
  1. ajouter une déclaration `display: none;` dans cette règle
  2. si l'élément ne disparaît pas, c'est probablement le sélecteur qui est incorrect !

Plan

Moteur d'une interface graphique



Modèle évènementiel

Avantages

- boucle d'attente d'évènements simultanément à l'écoute de plusieurs sources (clavier, souris, ...)
- programmation plus simple et concise qu'une boucle de scrutation

Contraintes

- les fonctions de réaction doivent s'exécuter rapidement
  - pas de calcul lourd
  - pas de temporisation

  dans le cas contraire on gèle l'interface
- l'exécution est non séquentielle, difficile à comprendre
- il faut partager des objets et maintenir un état pour communiquer entre fonctions de réaction

Dans une page HTML

```
<element evenement="code_a_executer">
```

- Le mot clé `this` désigne l'élément qui a déclenché l'événement.
  On écrit donc souvent
  `<element evenement="une_fonction(this);">`
- Pour déboguer : `console.log(...);` écrit dans la console des outils web.
  On peut lui donner des variables en argument.
- Le code à exécuter est écrit en JavaScript, la plupart du temps stocké dans un fichier `.js` à part

Plan

Éléments de syntaxe

```
function calcul(n) {
  let prec = 0;
  let result = 0;
  for (let i = 2; i <= n; i += 1) {
    if (i <= 10) {
      result = prec + result + i;
      prec = result;
    }
  }
  return result;
}
```

- `for` et `if` similaires à C ou Java
- `let` pour déclarer les variables (et définir leur portée)
- fonctions non typées
- accolades `{ }` pour délimiter les blocs
  et point-virgule `;` pour séparer les instructions

Quelques facilités supplémentaires

- boucles for pour parcourir les listes :

  ```
  for (x of liste) { ... }
  ```

  Attention : `for (x in objet)` existe aussi mais il parcourt les attributs de l'objet.
- chaînes de caractères : un peu comme en Python
  - délimitées par ' ' ou " "
  - concaténation +

Plan

Document Objet Model

La page HTML est vue comme un objet manipulable par JavaScript :

```
function replace_btn() {
  this.innerHTML = prompt('Saisir une valeur');
}

document.getElementById('btn-exec').addEventListener(
    'click', replace_btn);
```

Voir aussi

`https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model`

Chaque élément de la page est un nœud du DOM :

- les éléments contenus dans un élément sont ses fils
- les attributs d'un élément sont des attributs de l'objet JavaScript correspondant
- `element.innerHTML` donne accès (en lecture ET écriture) à tout le contenu de l'élément

Accéder à un élément particulier

- `document.getElementById(identifiant)` :
  l'élément portant l'identifiant donné
- `document.getElementsByTagName(balise)` :
  la liste des éléments ayant la balise donnée
- `document.getElementsByClassName(classe)` :
  la liste des éléments portant la classe donnée
- `document.querySelectorAll(selecteur)` :
  la liste des éléments correspondant au sélecteur CSS donné

Génération procédurale de la page

```
function create_heading() {
  let heading = document.createElement("h1");
  let heading_text = document.createTextNode("Big Head!");
  heading.appendChild(heading_text);
  document.body.appendChild(heading);
}

document.addEventListener("DOMContentLoaded",
                          create_heading);
```