Formulaires HTML

Alexandre Niveau

GREYC — Université de Caen

Retour sur la méthode GET

- Pour l'instant, on n'a utilisé qu'une seule méthode de HTTP : la méthode `GET`, qui permet de *récupérer* une ressource
- Caractéristique importante : c'est une « *safe method* » ⇒ elle est définie comme devant n'avoir **aucun effet de bord**

  ➤ elle n'est pas censée modifier l'état de l'application (hormis pour des détails secondaires, comme un compteur de visites)

  ➤ elle est « sûre », au sens où on peut l'utiliser sans crainte ⇒ important par exemple pour les robots indexeurs !
- On peut donner des paramètres à l'URL demandée avec `GET`, qui vont affecter la façon dont la ressource est récupérée, ou le type de contenu
- C'est le standard (RFC 7231) qui impose que GET soit une safe method, mais en pratique il n'y a aucune contrainte technique qui oblige à respecter cette contrainte

  ➤ il est *possible* d'utiliser les paramètres d'URL pour faire des *actions* (qui modifient l'état) sur un site web… mais c'est une très mauvaise idée !
- Pour faire des actions, il est préférable d'utiliser d'autres méthodes

Les méthodes non *safe*

- Il y a trois méthodes HTTP qui permettent de modifier l'état de l'application :
  - POST permet d'envoyer des données
  - PUT permet d'enregistrer une ressource à un chemin donné
  - DELETE permet de supprimer la ressource d'un chemin donné
- En pratique les deux dernières sont rarement utilisées (voir plus loin), c'est en général POST qui est utilisée pour tous les cas de figure non couverts par GET

Anatomie d'un POST

- Pour envoyer des données avec la méthode `POST` de HTTP, on les met dans le corps de la requête
- Le format des données est libre, mais en général il suit la même syntaxe que les paramètres d'URL (à quelques détails près)

  ➤ formellement, il s'agit du format MIME `application/x-www-form-urlencoded`

```
POST /chemin/pagederecup HTTP/1.1
```

```
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 21

nom=Toto&couleur=vert+clair
```

- Couples clef-valeur séparés par des `&`
- Espaces remplacées par `+`
- Encodage particulier des caractères spéciaux et non-ASCII

Formulaires HTML

- HTML fournit une interface permettant aux internautes de construire « interactivement » ces couples clef-valeur : les *formulaires*
- Ils sont principalement utiles pour envoyer des données (méthode `POST`) : poster un commentaire, s'authentifier sur un site…
- Mais ils permettent aussi de construire une URL paramétrée : champ de recherche, contrôle de l'affichage du contenu (ordre, filtrage)…
- Ils fonctionnent exactement de la même manière pour les deux usages ; dans les deux cas le navigateur utilise le contenu du formulaire pour construire une chaîne de couples clef-valeur
- Il faut juste préciser une URL et la méthode à utiliser :
  - si on met `POST`, le navigateur met la chaîne dans une requête `POST` qu'il envoie à l'URL fournie
  - si on met `GET`, le navigateur met la chaîne à la fin de l'URL fournie après un `?`, et il envoie une requête `GET` à l'URL paramétrée ainsi obtenue

Base d'un formulaire

- L'élément représentant un formulaire est `form`

```
<form action="page_de_recup" method="post">
    …
</form>
```

- Attribut `action` : URL de la page qui va récupérer les données (par défaut, la même page)
- Attribut `method` : méthode HTTP à utiliser pour transmettre les données ; soit `GET` soit `POST` (par défaut `GET`)

Élément `input`

- Les formulaires sont constitués de texte et de widgets
- La plupart des widgets sont créés avec l'élément `input`
- Le type de widget est contrôlé par l'attribut `type`

```
<form>
   <input type="text" />
   <input type="checkbox" />
```

```
   <input type="radio" />
</form>
```

Labels

- Les légendes des champs du formulaires doivent être placés dans des éléments `label` (étiquettes)
- Les labels doivent être explicitement associés à leurs champs : *extrêmement important* pour l'accessibilité et pour l'ergonomie !
- Pour associer un label à un champ, le plus simple est de mettre le champ *dans* le label :

```
<form>
  <label>Ville : <input type="text" /></label>
  <label>Capitale : <input type="checkbox" /></label>
</form>
```

(Cliquer sur les labels !)

Autre solution pour associer un label à un champ

- Si on ne veut pas mettre le champ dans le label, on peut aussi relier le label au champ grâce à l'attribut `for`, qui fait référence à l'`id` du champ. *En général, la solution précédente suffit, est plus rapide à écrire et donne un code plus lisible.*

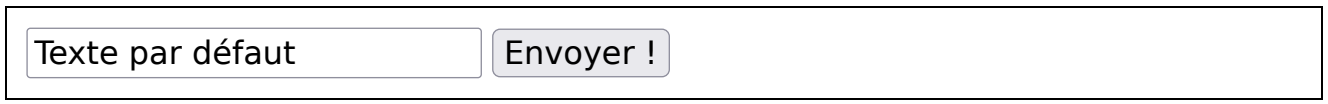
```
<form>
  <div>
    <label for="ville">Ville :</label>
    <input type="text" id="ville" />
  </div>
  <div>
    <label for="cap">Capitale</label>
    <input type="checkbox" id="cap" />
  </div>
</form>
```

(Cliquer sur les labels !)

Bouton de soumission

- Un formulaire est inutile si on ne peut pas le soumettre
- Élément `button`

```
<form>
   <input type="text" value="Texte par défaut" />
   <button type="submit">Envoyer !</button>
</form>
```

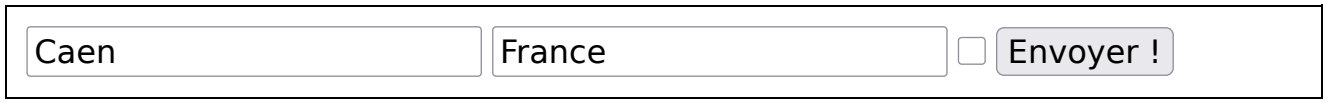


- Trois types de bouton :
  - `submit` envoie les données
  - `reset` remet tous les champs du formulaire à leur valeur par défaut
  - `button` ne fait rien ! Utile pour les scripts côté client

Noms des paramètres

- Les données entrées sur les widgets servent à donner des valeurs à des variables
- Il faut donner des noms à ces variables, pour que le serveur sache à quoi sert chaque donnée !
- La plupart des widgets utilisent l'attribut `name` pour cela

```
<form>
   <input type="text" value="Caen" name="ville" />
   <input type="text" value="France" name="pays" />
   <input type="checkbox" name="coche" />
   <button type="submit">Envoyer !</button>
</form>
```



- Attention au comportement des *checkbox*
- **Ne pas confondre les attributs `name` et `id`...**

Utilisation des données comme paramètres d'une page

- Dans l'exemple précédent, cliquer sur le bouton recharge la même page, mais l'URL a changé car le navigateur a ajouté des *paramètres d'URL*, construits à partir des noms des widgets et des valeurs fournies par l'internaute
- Pour renvoyer vers une autre page, il faut donner l'URL dans l'attribut `action` de l'élément `form`
- Dans cet exemple on renvoie vers une page qui fait quelque chose avec les paramètres :

```
<form action="demo/recup.php">
   <input type="text" value="Caen" name="ville" />
   <input type="text" value="France" name="pays" />
   <input type="checkbox" name="coche" />
```

```
   <button type="submit">Envoyer !</button>
</form>
```

Caen France Envoyer !

- NB : une simple page HTML ne sait rien faire avec les paramètres d'URL, c'est le serveur qui va analyser les paramètres et modifier le HTML envoyé en fonction.

Envoi des données

- Les paramètres d'URL ne conviennent pas à toutes les situations
- On a souvent besoin d'*envoyer* des données au serveur
- Dans ce cas, on peut spécifier que le navigateur doit utiliser la méthode `POST` de HTTP, en écrivant `method="POST"` comme attribut de l'élément `form` (la valeur par défaut de `method` est `GET` ⇒ construit une URL paramétrée)

```
<form action="demo/recup.php" method="POST">
   <input type="text" value="Caen" name="ville" />
   <input type="text" value="France" name="pays" />
   <input type="checkbox" name="coche" />
   <button type="submit">Envoyer !</button>
</form>
```

Caen France Envoyer !

GET vs POST

- Dans les deux cas, cliquer sur le bouton `submit` construit les couples clef-valeur et les communique au serveur
- N'importe quel programme peut être utilisé pour les récupérer : il suffit qu'il comprenne le protocole HTTP, ou qu'il puisse être lancé par le serveur lorsqu'un client demande l'URL en question (ex. : script PHP)
- Caractéristiques des deux techniques :
  - paramètres d'URL (`method="GET"`) : limités en pratique à environ 2000 caractères (voire 255) ; ne devraient être utilisés que pour filtrer et rechercher
  - corps de la requête HTTP (`method="POST"`) : pas de limite de taille, et pas visible par l'internaute
- NB : il est parfaitement possible d'envoyer des données avec `POST` à une URL paramétrée !
- Si hésitation entre paramètres d'URL et méthode POST, se demander si l'URL paramétrée aurait un sens en tant que lien. Il n'est *envisageable* d'utiliser la méthode GET que si la réponse est oui.
- **Attention** : les données de la requête `POST` ne sont pas *affichées* par les navigateurs, mais elles ne sont pas *cachées* !

  - Il est possible de les voir (par ex. avec l'outil « network monitor » de Firefox)
  - Elles circulent en clair sur le réseau

➤ Ne pas penser que `POST` est plus sécurisé que `GET`

Remarque sur les paramètres

- Les paramètres d'URL sont souvent utilisés abusivement pour le routage (mauvaises pratiques qui se sont répandues notamment par le biais de PHP), par ex. `http://toto.fr/forum?post=754`
- *Ce n'est pas non plus leur rôle* !

  ➤ L'URL du post 754 devrait être quelque chose comme `http://toto.fr/forum/posts/754`
- Les paramètres, comme leur nom l'indique, servent à *paramétrer* une page

  ➤ ils ne devraient être utilisés que pour *filtrer* ou *rechercher* :

```
http://toto.fr/forum?date=2015-02-23&sort=title
http://toto.fr/forum?q=glace+vanille
```

Boutons radio

- Les boutons radio n'ont d'intérêt qu'en groupe : un seul peut être sélectionné
- C'est l'attribut `name` qui les groupe
- Le paramètre prendra la valeur du bouton sélectionné : cette valeur correspond à l'attribut `value`

```
<form action="demo/recup.php" method="POST">
  Les formulaires HTML, vous êtes :
  <label><input type="radio" value="oui" name="fan" /> Fan</label>
  <label><input type="radio" value="non" name="fan" /> Pas fan</label>
  <label><input type="radio" value="?" name="fan" /> NSPP</label>
  <button type=submit>Envoyer !</button>
</form>
```

Les formulaires HTML, vous êtes : ○ Fan ○ Pas fan ○ NSPP Envoyer !

Grouper des champs

- L'élément `fieldset` permet de grouper des champs qui ont un rapport logique entre eux et de leur associer un label commun avec `legend`
- Utile notamment pour les cases à cocher et les boutons radio

```
<form action="demo/recup.php" method="POST">
    <fieldset>
      <legend>Les formulaires HTML, vous êtes :</legend>
      <label><input type="radio" value="oui" name="fan" /> Fan</label>
      <label><input type="radio" value="non" name="fan" /> Pas fan</label>
      <label><input type="radio" value="?" name="fan" /> NSPP</label>
    </fieldset>
```

```
    <button type=submit>Envoyer !</button>
</form>
```

Les formulaires HTML, vous êtes :
◯ Fan ◯ Pas fan ◯ NSPP

Envoyer !

Texte multi-lignes

- On utilise l'élément `textarea` pour du texte sur plusieurs lignes

```
<form action="demo/recup.php" method="POST">
  <textarea cols="30" rows="5" name="multi">Mettre du texte ici !</textare
  <button type="submit">Go</button>
</form>
```

Mettre du texte ici !

Go

- Propriété CSS `resize` pour que l'internaute puisse modifier la taille (valeurs : `horizontal`, `vertical`, `both`)
- Attention, contrairement à `input`, `textarea` a une balise ouvrante et une balise fermante (le contenu est utilisé comme valeur par défaut)

Valeur par défaut et *placeholder*

- Dans les champs texte, on peut mettre une valeur par défaut :

```
<form action="demo/recup.php" method="POST">
  <input type="text" size="10" name="prenom" value="Prénc
  <textarea cols="10" rows="5" name="comm">Votre commenta
  <button type="submit">Envoyer</button>
</form>
```

Prénom

Votre commentaire ...

Envoyer

- L'internaute doit effacer le contenu avant de taper
- En règle générale, c'est plutôt une sorte de « mode d'emploi » qu'on voudrait
- Il faut utiliser l'attribut `placeholder` :

```
<form action="demo/recup.php" method="POST">
  <input type="text" size="10" name="prenom" placeholder=
  <textarea cols="10" rows="5" name="comm" placeholder="V
  <button type="submit">Envoyer</button>
</form>
```

Prénom

Votre commentaire ...

Envoyer

- Attention, les exemples ci-dessous sont simplifiés : dans la pratique, il ne faut jamais utiliser un *placeholder* **à la place** d'un `label` ! Le formulaire doit être compréhensible sans le *placeholder*, qui n'est pas toujours affiché.

Menu déroulant

- Le menu déroulant s'obtient avec l'élément `select`, qui contient plusieurs éléments `option`

```
<form action="demo/recup.php" method="GET">
  <select name="fruit">
    <option>Fraise</option>
    <option>Pomme</option>
    <option>Raisin</option>
  </select>
  <button type="submit">Go</button>
</form>
```

Fraise Go

- La valeur du paramètre est le contenu de l'élément `option` sélectionné, ou le contenu de son attribut `value` s'il existe

Groupes d'options dans un menu

- On peut grouper les options d'un menu déroulant avec l'élément `optgroup`

```
<form action="demo/recup.php" method="GET">
  <select name="instrument">
    <optgroup label="Bois">
      <option value="clarinette">Clarinette (si♭)</option>
      <option value="basson">Basson (ut)</option>
      <option value="sax">Saxophone (mi♭)</option>
    </optgroup>
    <optgroup label="Cuivres">
      <option value="trompette">Trompette (si♭)</option>
      <option value="trombone">Trombone (ut)</option>
      <option value="tuba">Tuba (fa)</option>
    </optgroup>
  </select>
  <button type="submit">Go</button>
</form>
```

Clarinette (si♭) Go

- Améliore aussi l'accessibilité

Menu à choix multiples

- Choix multiples possibles en utilisant un menu déroulant `select` avec l'attribut

booléen `multiple`

```
<form action="demo/recup.php" method="GET">
  <select multiple name="fruit">
    <option>Fraise</option> <option>Pomme</option>
  </select>
  <button type="submit">Go</button>
</form>
```

- NB : attribut *booléen* = pas besoin de valeur, la simple *présence* de l'attribut lui donne la valeur « vrai »
- Si on teste l'exemple précédent, on voit que le paramètre est défini plusieurs fois dans l'URL. Il faut que le script de récupération sache quoi faire. Avec PHP, le plus simple est d'utiliser une syntaxe particulière pour le nom :

```
<form action="demo/recup.php" method="GET">
  <select multiple name="fruit[]">
    <option>Fraise</option> <option>Pomme</option>
  </select>
  <button type="submit">Go</button>
</form>
```

Autres widgets

- Il existe un certain nombre de `input` différents, notamment des variantes du champ `text` qui donnent plus de *sémantique* : `email`, `url`, `number`...
- Intérêt : éviter les erreurs de l'internaute (on ne peut pas soumettre le formulaire si les valeurs sont incohérentes) et augmenter l'ergonomie (par exemple les claviers proposés sur mobile peuvent être adaptés au champ)
- Voir par exemple ce guide en français sur Alsacréations [https://www.alsacreations.com/tuto/lire/1372-formulaires-html5-nouveaux-types-champs-input.html]

Quelques compléments

- Attribut `target` de l'élément `form` : indique où sera affichée la réponse. `_self` pour la même fenêtre (ou iframe), `_blank` pour une nouvelle
- Attribut booléen `checked` : permet qu'une case soit cochée par défaut, ou qu'un bouton radio soit sélectionné par défaut.
- Attribut booléen `selected` : idem, mais pour une option dans un menu déroulant.

Validation côté client

- Attribut booléen `disabled` : valable pour tous les widgets, permet de les rendre non modifiables.
- Attribut booléen `required` : valable pour tous les widgets, permet de les rendre obligatoires.

- Voir le tutoriel sur MDN [https://developer.mozilla.org/fr/docs/Learn/Forms/Form_validation] pour d'autres compléments sur la validation des formulaires (attributs HTML `pattern`, `minlength` et `maxlength`, `min` et `max`, etc., pour empêcher l'internaute de soumettre des valeurs aberrantes)

Style des formulaires

- Tout n'est pas modifiable par CSS dans les formulaires, notamment les menus déroulants…
- Pour sélectionner un type d'`input` : utiliser les sélecteurs par attribut

```
input[type=text] {
        border: 2px dotted green;
}
```

- Nombreuses pseudo-classes utiles pour les éléments de formulaire
  - `:focus`, sélectionne les éléments qui ont le focus, très utile pour les champs texte
  - `:checked` sélectionne les cases cochées, les boutons radio sélectionnés, et les options choisies dans un menu déroulant (peut être détourné, un peu comme `:target`, pour rendre les pages un peu dynamiques !)
  - `:default` sélectionne les éléments qui sont dans leur état par défaut (ex.: cases à cocher)
  - `:valid` et `:invalid`, pour styler les champs dont la validation est correcte ou non
  - `:disabled` et `:enabled` sélectionnent les éléments désactivés ou non par l'attribut HTML `disabled`
  - `:required` et `:optional` sélectionnent les élements obligatoires ou non (attribut HTML `required`)
- Attention à `width` : les widgets ont chacun leur propre interprétation. Préciser `box-sizing: border-box;` pour être tranquille
- Tous les détails sur le tutoriel de MDN [https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/Styling_HTML_forms]

Spécifications et normes

- W3C : HTML5 Forms [http://www.w3.org/TR/html5/forms.html]

Tutoriels

- Formulaires HTML5 : nouveaux types de champs [https://www.alsacreations.com/tuto/lire/1372-formulaires-html5-nouveaux-types-champs-input.html] sur Alsacréations
- HTML forms guide, [https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms] tutoriel très complet de MDN
- Creating Accessible Forms [http://webaim.org/techniques/forms/]
- Form Layout with CSS [http://www.gargan.org/en/Web_Development/

Form_Layout_with_CSS/]

 [http://creativecommons.org/licenses/by/4.0/]