Architecture(s) et application(s) Web

Télécom SudParis Olivier Berger (TSP)

28/08/2025

CSC4101 - Interface dynamique côté navigateur

Table des matières

Objectifs de cette séquence

Cette séance introduit les mécanismes permettant la mise en œuvre de certaines fonctionnalités des applications Web du **côté du client**, dans le navigateur Web.
On présente les technologies JavaScript comme JQuery, qui permettent par exemple de rendre l'interface des applications plus dynamique et d'améliorer l'expérience des utilisateurs.
Jusqu'ici, l'interface des applications Web est basée sur le contenu de pages HTML produites sur le serveur, et visualisée par le navigateur.
On introduit ici les mécanismes permettant de faire tourner du côté du client Web, dans le navigateur, du code applicatif qui servira à **modifier le contenu de l'arbre DOM** de ces pages.
L'utilisateur peut donc percevoir une **interface modifiée dynamiquement**, de façon indépendante du serveur d'origine, en fonction de ses **interactions avec le navigateur** (mouvements souris, etc.).
Ce code exécuté sur le navigateur peut inclure le chargement de ressources additionnelles depuis la Toile, permettant ainsi d'intéragir avec de multiples serveurs ou de réaliser des agrégations de ressources, comme cela a été popularisé par l'émergence du Web 3.0 avec l'approche AJAX.

1 Aller au-delà des formulaires basiques Symfony

Cette section vise à montrer les limites des formulaires « basiques » tels qu'ils sont générés par l'assistant de Symfony `make:crud`, et à voir comment obtenir une expérience utilisateur améliorée, notamment pour les champs des attributs multi-valués.

1.1 Edition des OneToMany

Améliorons l'UX des formulaires…
Rappel modèle données :
— `Project` et `Todo` reliées par un *OneToMany*

1.2 Version de base

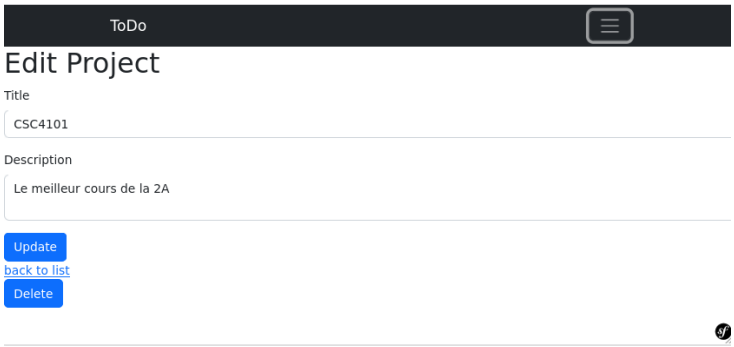


Figure 1 – version du formulaire d'édition généré par `make:crud`

1.3 Ajout d'une association OneToMany

Propriété *multi-valuée* `Project::todos`

```php
class ProjectType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('description')
            ->add('todos')
            ;
```

Listing 1 : *form builder* pour `Project` dans `src/Form/ProjectType.php`

1.4 Champ par défaut : liste à sélection multiple

Problème : comment sélectionner plusieurs valeurs ? (*Crtl* + *clic*)

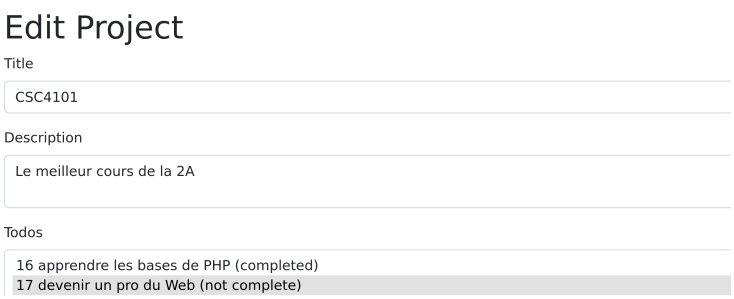


Figure 2 – version basique avec liste

1.5 Alternative : utilisation de *checkboxes*

On peut remplacer le widget associé à la liste (modifiable) des `todos` par une série de cases à cocher (*checkboxes*).

```
class ProjectType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('description')
            ->add('todos', null, [
                            'multiple' => true,
                            'expanded' => true
                        ])
            ;
```

Par défaut, génère un champs *ChoiceType* … OK, merci la doc…

Si on ne précise pas le type du champ (second argument à `null`), comme il s'agit d'une propriété multi-valuée (côté *many* d'une OneToMany), Symfony génère un champ de formulaire de type `ChoiceType`.
Cf. explication des options du type de champ ChoiceType : multiple et expanded.

1.6 Résultat : propriété multi-valuée via des *checkboxes*

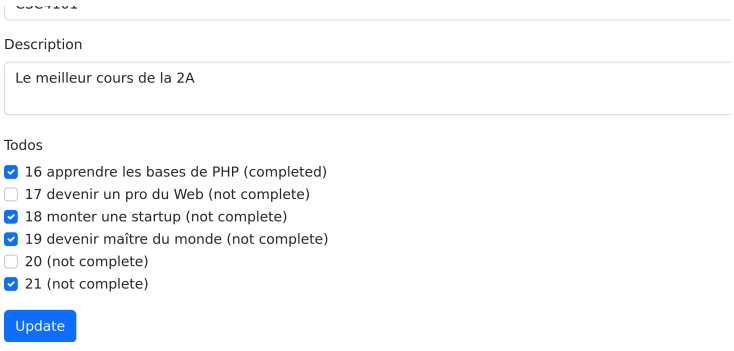


Figure 3 – sélection multiple via *checkboxes*

1.7 Encore faut-il réparer la sauvegarde

`'by_reference' => false` nécessaire pour que ça sauvegarde les modifications sur les tâches du projet … merci la doc…

```php
class ProjectType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('title')
            ->add('description')
            ->add('todos', null, [
              'multiple' => true,
              'expanded' => true,
              'by_reference' => false
          ])
            ;
```

Listing 2 : correction pour permettre la sauvegarde

1.8 Mise en forme du formulaire

1.8.1 Comment faire un affichage en colonnes ?

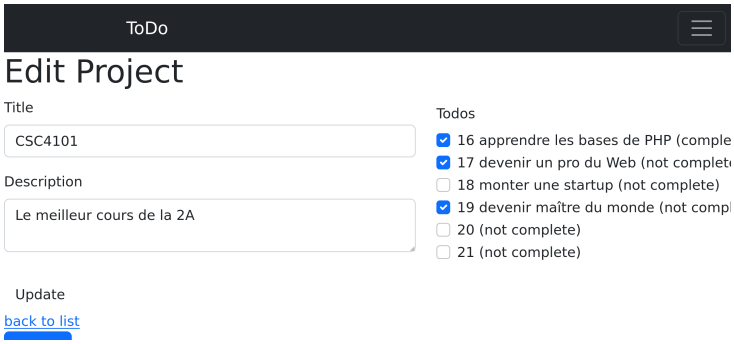


Figure 4 – séparation des champs en plusieurs colonnes

On souhaiterait améliorer encore la présentation pour la rentre plus compacte, sur deux colonnes.

1.8.2 Modifier les gabarits en profondeur

Rappel : gabarit `project/edit.html.twig` généré par défaut par `make:crud`
`make:crud` a généré plusieurs éléments de gabarits qui s'incluent à tour de rôle.
Commençons par le gabarit `project/edit.html.twig` :

```
{% extends 'base.html.twig' %}

{% block title %}Edit Project{% endblock %}

{% block body %}
    <h1>Edit Project</h1>

    {{ include('project/_form.html.twig', {'button_label': 'Update'}) }}

    <a href="{{ path('app_project_index') }}">back to list</a>

    {{ include('project/_delete_form.html.twig') }}
{% endblock %}
```

Sous-gabarit pour le formulaire `project/_form.html.twig`

```
{{ form_start(form) }}
```

```
    {{ form_widget(form) }}
    <button class="btn btn-primary">{{ button_label|default('Save') }}</button>
{{ form_end(form) }}
```

Éclater la structure des *widgets* du formulaire

```
{{ form_start(form) }}
    {% dump form.todos %}
    {{ form_errors(form) }}
    <div class="row">
        <div class="col">
            {{ form_row(form.title) }}
            {{ form_row(form.description) }}
        </div>
        <div class="col">

            {{ form_row(form.todos) }}
        </div>
    </div>
    <button class="btn">{{ button_label|default('Save') }}</button>
{{ form_end(form) }}
```

Cf. documentation Symfony How to Customize Form Rendering pour savoir comment remplacer `{{ form_widget(form) }}` par du code HTML plus spécifique.

1.8.3 Améliorer l'affichage des *checkboxes*

Figure 5 – affichage des tâches en HTML stylé

Surcharge du gabarit par défaut… c'est faisable, mais à quel coût…

```
{% extends 'base.html.twig' %}

{% form_theme form _self %}
{% block _project_todos_entry_widget %}
  {% set entity = form.parent.vars.choices[form.vars.value].data %}
  {% set checked = form.parent.children[form.vars.value].vars.checked %}
  <div class="form-check">
    <input type="checkbox" id="project_todos_{{ entity.id }}"
           name="project[todos][]" class="form-check-input"
           value="{{ entity.id }}"
       {% if checked %}
           checked="checked"
       {% endif %}
     />
    <label class="form-check-label" for="project_todos_{{ entity.id }}">
      {% if entity.title is empty %}(<i>unnamed todo #{{ entity.id }}</i>)
      {% else %}{{ entity.title }}
```

```
      {% endif %}
    </label>
  </div>
{% endblock %} {# _project_todos_entry_widget #}

{% block title %}Edit Project{% endblock %}

...
```

Cf. documentation Symfony How to Work with Form Themes pour savoir comment surcharger des éléments de construction des fragments de gabarits, qui génèrent le code HTML pour les champs pour les collections.
Ça fonctionne, mais probablement risqué si Symfony évolue et qu'on obtient quelque chose de difficilement maintenable

1.9 Et pour le projet ?

Faites-en sorte que ça fonctionne… mais on n'est pas là pour faire une vraie application, donc ne pas faire tout cela.
Des listes à sélection multiple peuvent suffire… même si *Ctrl+clic* ce n'est pas formidable.
Sinon, explosion de l'effort nécessaire.

2 Interfaces dynamiques

Cette section présente le principe des interfaces Web dynamiques, où les pages visualisées par le navigateur Web sont modifiées directement sur le navigateur, via des programmes en JavaScript.

2.1 Vues générées côté serveur

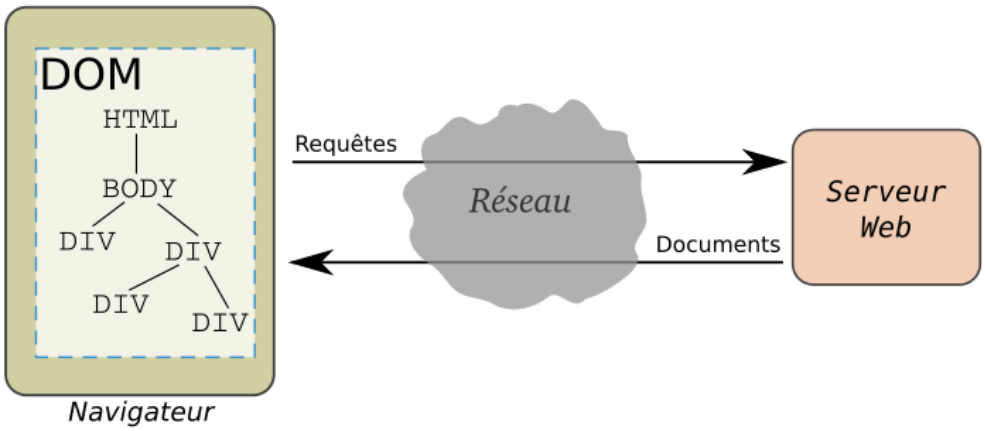


Figure 6 – Documents statiques

Jusqu'à présent, nous avons travaillé sur un modèle d'application dont les vues reposent sur des pages HTML qui sont intégralement calculées côté serveur et où la grande majorité des interactions sont gérées par le Contrôleur dont le code est côté serveur.
Seul le CSS introduit des éléments de variabilité côté client, mais le HTML reste inchangé.

2.2 Vues générées côté client

« HTML dynamique »

On peut améliorer le modèle précédent pour ajouter des éléments de programme du côté client, qui permettent de faire évoluer la vue et gérer d'autres interactions, sans avoir besoin de refaire appel à une requête auprès du serveur.
Cela introduit moins de latence liée aux transferts réseau, ou de charge sur le serveur.
Principalement utilisé pour améliorer l'expérence utilisateur en mode GUI, traité au plus près des actions de l'utilisateur, sans recourir à HTTP.

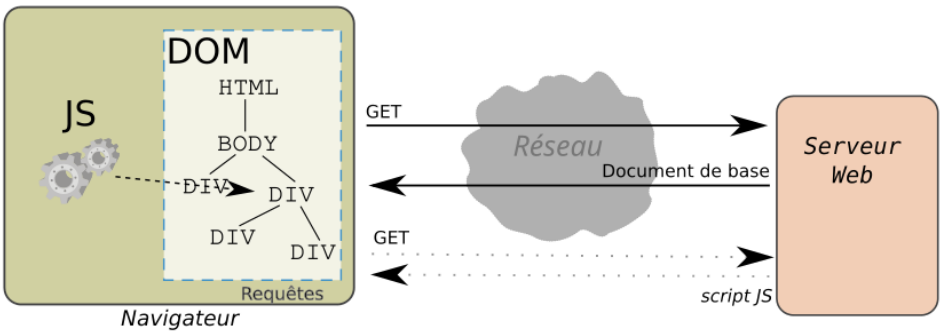


Figure 7 – Documents dynamiques avec Javascript

3 JavaScript

Cette section présente les caractéristiques du langage JavaScript

3.1 JavaScript pour applications Web

Abrégé : JS
- Langage de script orienté objet non typé
- Pilier dans la création d'applications Web
  - Principalement utilisé côté client : interprété par les navigateurs Web
  - Mais peut aussi être utilisé côté serveur (exemple *NodeJS*)

Aujourd'hui, JavaScript peut être utilisé pour programmer le *backend* côté serveur, par exemple avec NodeJS, en lieu et place de PHP.
Dans ce cours, et dans votre projet, vous utiliserez cependant Javascript uniquement pour l'exécution dans le navigateur.

3.2 Intérêt d'un *script* côté navigateur

- Améliorer les temps de réponse (exécution côté client, pas de latence d'envoi vers le serveur)
- Contrôler les données des formulaires avant envoi vers le serveur (éviter les envois erronés)
- Réaliser des pages animées
- Modifier le contenu, le style de la page courante en fonction des actions de l'utilisateur sur la page

3.3 Historique

- Développé en 1995 par Brendan Eich pour Netscape (nom d'origine *LiveScript*)
- Grand succès au début du Web (mais nombreuses incompatibilités entre navigateurs)
- Nombreuses étapes de standardisation nécessaires
  - ECMA (*European Computer Manufacturers Association*) a défini le standard ECMAScript
- L'arrivée d'**AJAX** avec le Web 2.0 replace JavaScript au premier plan (> 2005)

JavaScript devient même populaire hors du Web. Ex. programmes pour l'environnement de bureau *Gnome Shell*

3.4 Principes de base

- Un **noyau** (le cœur du langage)
  - des opérateurs, des structures, des objets prédéfinis (`Date`, `Array`…)
- Un ensemble d'**objets associés au navigateur**
  - fenêtre (*window*), document (*document*), formulaire (*form*), image (*image*)…
- **Programmation événementielle**
  - Capter les évènements : clics, déplacement de la souris, chargement de la page …
  - Associer des fonctions aux événements

On retrouve un style de programmation évenementielle courant dans d'autres contextes, dès qu'il y a des interfaces graphiques

3.5 Plate-forme

— L'interface **DOM** (*Document Object Model*) standardise les méthodes pour accéder par objets aux documents
— Nombreuses bibliothèques (*frameworks*) aident à programmer.
Ex : Dojo, Rialto, **Angular**, **JQuery**, Yui…
— « *bytecode* » pour programmer à un plus haut niveau, dans un langage compilé en JS : TypeScript, …
— **JSON** (*JavaScript Object Notation*) utilise la notation des objets JavaScript pour transmettre de l'information structurée

```
{
  "nationalTeam":"Norway",
  "achievements":[
      "12 Olympic Medals",
      "9 World Championships",
      "Winningest Winter Olympian",
      "Greatest Nordic Skier"
     ],
  "name":"Bjoern Daehlie",
  "age":41,
  "gender":"Male"
}
```

De nouveaux langages, comme TypeScript ou Elm, introduisent des paradigmes de programmation plus sécurisés et améliorant la qualité du code (typage strict, fonctionnel, etc.), tout en restant compatibles avec la plate-forme ECMAScript, car compilés en JavaScript, et permettant donc un déploiement sur tout navigateur.

3.6 Utilisation

3.6.1 Intégration dans pages HTML

— Code JS contenu ou référencé dans balises `<script>` et `</script>`
  — Soit directement :

```
<head>
    <script>
        code javascript (fonctions, variables...)
    </script>
</head>
```

  — Soit dans un fichier séparé (ou URL externe) :

```
<head>
    <script src="monprog.js"></script>
</head>
```

3.6.2 Appel des fonctions JavaScript

— à partir d'évènements dans les balises

```
<input type="button"
       onClick="alert('clic sur bouton');">
```

— à partir d'un lien

```
<a href="javascript:affiche(menu);">
```

— par d'autres fonctions

```
function hello() {
  alert("hello");
}
```

3.7 Gestion d'événements

— Chaque événement a un traitement par défaut
— On peut associer une fonction à un évènement, pour l'exécuter **avant** traitement par défaut
  — Si retourne faux (*false*), alors traitement par défaut pas exécuté
  — Si retourne vrai (*true*), alors traitement par défaut exécuté ensuite
— Association fonction à évènement utilise un mot-clef de type « *onEvent* » (`onClick`, `onLoad`, `onChange`, `onMouseOver`, `onMouseOut`, `onKeyDown`…)

3.8 Exemples de réflexes

— clic sur bouton :

```
<input type="button" name="surface" value="Surface"
       onClick="Test(all);">
```

— déplacement souris au-dessus :

```
<a href="menu.html" onMouseOver="affiche();">
```

— fin de chargement du document :

```
<body onLoad="init(1);">
```

3.8.1 Exemples évènements / balises

Evénements	Balises	Fonctions
onBlur	*	désélection
onChange	*idem*	modification
onFocus	*idem*	sélection
onSelect	password, text, textarea	sélection de valeur
onClick	a href, button, checkbox, radio, reset, submit	clic
onMouseover	a href, area	souris pointe
onLoad	body	chargement page
onSubmit	form	soumission

Cf. liste complète d'événements

3.8.2 Exemple de gestion d'événements

```
<html>
    <head>
        <script language="JavaScript">
            function maFonction() {
                alert('Vous avez cliqué sur le bouton GO');
            }
            function autreFonction() {
                alert('Cliqué sur le lien "Cliquer ici"');
            }
        </script>
    </head>
    <body>
      <ol>
        <li>Appel sur évènement :
```

```
            <input type="button" value="GO" onClick="maFonction();">
        </li>
        <li>Appel sur un lien :
            <a href="javascript:autreFonction();">Cliquer ici</a>
        </li>
      </ol>
      <a href="javascript:history.go(-1);">Retour</a>
    </body>
</html>
```

3.9 Objets JavaScript

- JavaScript a des types primitifs et des objets
- Types primitifs créés par affectation directe
- Utilisation d'objets
  - Instanciation par un constructeur
  - Syntaxe *à la java* :
    - Accès aux attributs : `monObjet.attributs`
    - Accès aux méthodes : `monObjet.méthode()`
  - Modèle objet beaucoup moins strict que Java

3.9.1 Exemple

```
// Création d'une variable
var etienne = {
    nom : "etienne";
    branche : "marketing";
}

// Création d'un objet par constructeur
function Employe (nom,branche) {
    this.nom = nom;
    this.branche = branche;
    this.fullDescription = function () {
        return this.nom + " " + this.branche;
    };
}

var marc=new Employe("marc","informatique");

marc.fullDescription(); // appel de méthodes
```

3.9.2 Objets prédéfinis 1/2

Les objets du noyau
- `Object` : création d'un objet
- `Function` : création d'une fonction
- `Array` : création d'un tableau
- `String` : création d'une chaîne de caractères
- `Number` : création d'un nombre
- `Date` : création d'une date
- `Math` : création d'un objet mathématique
- etc.

3.9.3 Objets prédéfinis 2/2

Les objets du navigateur :
- fenêtre,
- document,
- formulaires
- ...

3.9.4 Hiérarchie d'objets présents en mémoire dans le navigateur

- Un navigateur (objet *navigator*),
  - Contient une fenêtre (*window*),
    - Qui contient son document HTML (*document*),
      - ses formulaires (*form[]*), ses images (*image[]*), ses liens (*link[]*)…
- Un formulaire contient
  - des boutons (*button[]*),
  - des listes déroulantes (*select[]*)
    - qui contiennent des options (*option[]*)
- etc.

3.9.5 Attributs et méthodes

- Les attributs d'un objet correspondent aux attributs des balises HTML (par exemple *height*, *width*, *name*, *src* … pour l'objet *image*)
- Les méthodes correspondent aux fonctionnalités du navigateur (par exemple *back*, *forward*, *go* pour l'historique (*history*))
- Certains attributs et méthodes sont communs à un ensemble d'éléments

Se rapporter aux références pour connaître les méthodes et attributs d'un élément, par exemple Référence W3School

3.10 Mise au point

Délicate avec les langages interprétés car :
- Les erreurs n'apparaissent qu'à l'éxécution
- Par défaut, les erreurs ne sont pas affichées

3.10.1 Outil : console JS

- **console de développement** du navigateur
  - Possibilités variables selon les navigateurs
  - Indication des erreurs de syntaxe
  - Inspection du contenu des variables
  - Débogueur, points d'arrêt, affichage des variables
- Affichage d'informations sur la console :
  - Affichage des erreurs de syntaxe éventuelles
  - Par programme avec la fonction *consoleLog(« message » + variable)*

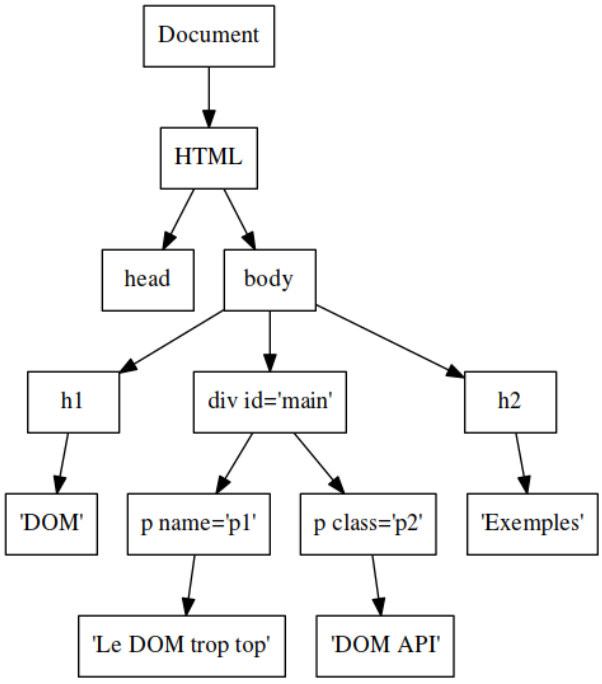
4 Manipulation du DOM

Cette section rappelle le principe du *Document Object Model* (DOM), déjà vu en lien avec CSS.
Cette fois, on s'intéresse surtout à l'interface de programmation standardisée de manipulation du DOM.
Elle permet aux programmes de manipuler le contenu des pages Web en JavaScript, de façon portable quelque soit le navigateur.

4.1 DOM

Document Object Model (Modèle Objet du Document)
- Modèle de la structure des documents HTML
- Interface de programmation standardisée par le W3C (API DOM)
- Implémentée par les navigateurs avec plus ou moins d'extensions
- Références :
  - spécifications du W3C,
  - le DOM Gecko (Mozilla)

4.1.1 Exemple d'arbre DOM




```
<html>
  <head>
    ...
  </head>
  <body>
    <h1>DOM</h1>
    <div id='main'>
      <p name="p1">Le DOM trop top</p>
      <p class="p2">DOM API</p>
    </div>
    <h2>Exemples</h2>
  </body>
</html>
```

4.2 Manipulation avec JavaScript

— Le DOM offre un ensemble de méthodes de parcours de l'arbre pour accéder et/ou modifier son contenu
  — ajouter des éléments
  — supprimer des éléments
  — ajouter et/ou modifier des attributs
  — etc.

4.3 Exemple d'utilisation du DOM

— Le DOM comporte de nombreuses méthodes pour accéder à un élément, de différentes façons :
  — par son *identifiant* : `<div id="intro">`
  — par sa *position* dans l'arbre : (fils, parent…)
  — par son *type* : ensemble des paragraphes, des niveaux `h2`, …

4.4 Utilité

— Modifier l'apparence
— Par exemple, changer dynamiquement le lien (attribut `class`) avec des définitions de feuilles de style : interfaces réactives
— Exemple : masquage, démasquage des slides dans reveal.js, utilisé pour les diapos projetées en CM

Pour l'utilisateur, on dirait que des actions graphiques sont effectuées à l'écran, mais en fait, le code JavaScript effectue des modifications sur l'arbre DOM, et le moteur de rendu du navigateur redessine la page, en appliquant le CSS, ce qui crée les différences d'affichage.
JavaScript permet également de faire des choses au niveau de tout ce que sait faire le navigateur, en vrai langage de programmation, indépendant d'une utilité pour le Web.
On peut ainsi aller très loin, pour réaliser des choses très pointues : émuler un système d'exploitation, virtualisé dans le navigateur, etc.

5 Framework JavaScript jQuery

Cette section présente le *framework* de développement JQuery qui facilite le développement en JavaScript.

5.1 Avantages d'un *FrameWork*

Faciliter la tâche des développeurs.
- Il existe un grand nombre de cadriciels (*frameworks*) JS
- Lequel choisir ? éviter d'en utiliser plusieurs dans une même aplication
- Critères de choix : les performances, les fonctionnalités, le poids, la commmunauté, etc.
- Il existe une page de comparaison sur Wikipédia : `https://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_web_frameworks`

5.2 jQuery

`https://jquery.com/`
- Bibliothèque des plus populaires et des plus utilisées
- Utilisable avec tous les navigateurs pour un résultat identique (était utile avant la standardisation de JS)

Un peu ancien, et il y a sûrement mieux aujourd'hui… mais montré ici pour illustrer, pas comme recommendation ultime.

5.2.1 Caractéristiques

- Slogan : « *write less, do more* » : écrire moins pour faire plus
- Facilité d'apprentissage
- Simplification de la programmation JavaScript
- Encapsulation des actions courantes dans des méthodes, ce qui réduit le code à écrire
- Simplification de certaines tâches comme la manipulation du DOM ou la programmation AJAX

5.2.2 Fonctionnalités

- Accéder aux objets HTML/DOM
- Manipuler les styles CSS
- Gérer les évènements
- Réaliser des effets et des animations
- Programmer avec AJAX
- etc

jQuery dispose aussi d'extensions sous forme de *plugins*

5.2.3 Chargement de JQuery

- téléchargé depuis `jQuery.com` et inclus en local sur notre site

```
<head>
  <script src="jquery-3.7.1.min.js"></script>
</head>
```

— ou référencer sur serveur de contenu externe, p. ex. sur CDN (*Content Delivery Network*) jQuery

```
<head>
  <script src="https://code.jquery.com/jquery-3.7.1.min.js">
  </script>
</head>
```

Attention : traçage des utilisateurs

Remarque : il existe 2 versions : compressée (.min) ou non.

5.2.4 Programmation « événementielle »

1. Le navigateur (lui-même), ou les interactions de l'utilisateur déclenchent des **événements**
2. les événement déclenchent l'exécution d'actions sur les objets du programme (dont le DOM).

5.2.5 Structure générale des programmes

1. **Sélectionner** des éléments du DOM HTML : générer des objets javascript qui les représentent
2. Installer des gestionnaires d'événements / **actions** sur ces éléments : ces gestionnaires sont des **fonctions**

5.2.6 Syntaxe de base

— Syntaxe de base jQuery :

```
$(selector).action(...)
```

— La construction `$()` permet d'implanter un traitement avec jQuery
— Le sélecteur (`selector`) permet de sélectionner un (ou plusieurs) élément(s) HTML
— L'action (`action`) est exécutée sur chaque élément sélectionné

5.2.7 Actions

— Actions immédiates : changer l'état d'un élément du DOM. Exemple :

```
$(this).hide() // cacher l'élément courant
```

— Mise en place de **réflexes** (*callbacks*) sur des événements (actions différées) :

```
$('#bouton1').click(function() {
    // actions à faire lors d'un clic sur le bouton
    //...
});
```

5.2.8 Point d'entrée du code

— Pour éviter que nos actions jQuery s'exécutent sur des objets non encore chargés, jQuery permet de ne démarrer le tout qu'une fois reçu l'évènement « *document ready* » :

```
$(document).ready(function(){

   // code jQuery...

});
```

5.2.9 Exemples sélecteurs JQuery

l'élément courant `$(this)`

le nom de la balise `$("p")`, pour tous éléments `<p>`

l'identifiant `$("#id1")` : l'élément ayant l'identifiant `id1`

la classe `$(".test")` tous éléments ayant attribut `class test`

Même syntaxe que les sélecteurs CSS

6 AJAX

Cette section présente le modèle « AJAX » qui permet de mettre en œuvre des applications JavaScript qui interagissent avec des services Web externes pour récupérer des données, de façon asynchrone.

6.1 AJAX

Asynchronous JavaScript And XML
- Ne plus lire l'acronyme littéralement (~~XML~~)
- AJAX signifie qu'on intègre différentes technologies
  - centrées sur Javascript
  - **requêtes HTTP asynchrones** vers serveurs
- Traitement de **données supplémentaires** récupérées sur serveur(s)
- Exemple : construction progressive d'un formulaire en fonction des réponses de l'utilisateur.

Terme introduit en 2005. Plus lié à la technologie XML aujourd'hui, mais plutôt à HTML et JSON, par exemple… mais le nom de la fonction n'a pas été changé

6.2 Classe `XMLHttpRequest` (XHR) de JavaScript

- Élément clef de la technologie AJAX
- Créé en 1999 par Microsoft comme objet ActiveX
- Intégré comme objet JavaScript par Mozilla.
- Création : `new XMLHttpRequest();`
- Ensuite, utilisation comme tout objet via ses
  - propriétés
  - et/ou ses méthodes
- Alternative avec jQuery (voir plus loin)

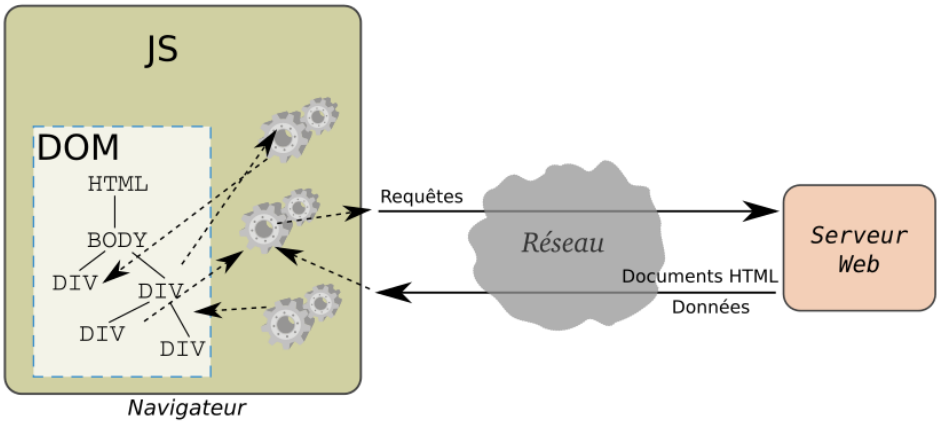
Encore une fois, pas spécifique à XML : on peut charger ce qu'on veut sur le serveur cible.

6.3 Principe de fonctionnement

- Un évènement (clic, saisie de texte…) provoque l'exécution d'une fonction JavaScript
- Cette fonction
  1. instancie un objet `XMLHttpRequest`
  2. transmet sa requête HTTP à un serveur (`GET` d'un document ou `POST` de données, ou …)
  3. récupère la réponse lorsque celle-ci est disponible (asynchrone)
  4. met à jour la partie concernée de la page courante (un champ de saisie, une division…)

Naturellement, la fonction JavaScript peut
- appliquer différents traitements à la réponse obtenue
- modifier des attributs de styles de type CSS pour certains objets du document HTML.
- etc

6.4 Sous-requêtes HTTP



Les possibilité sont très nombreuses : à partir d'une page principale HTML, renvoyée une fois pour toutes par le serveur, et accompagnée d'un programme Javascript, on va pouvoir faire tourner une application complexe, qui se comporte comme un client HTTP vis-à-vis de services Web divers, et agrégeant les données renvoyées par différents serveurs HTTP, les traitant et les visualisant.
Cette architecture est de plus en plus populaire.
Elle déporte la gestion des Vues et des Contrôleur de l'application très fortement du côté du client, en comparaison de l'architecture traditionnelle qu'on voit dans le modèle étudié en cours jusqu'ici.

6.5 Utilisation d'AJAX avec jQuery

- L'utilisation de jQuery permet de faciliter l'écriture de programmes AJAX avec la fonction :
  `$.ajax()`
- En paramètres la description de la requête (objet au format JSON) :
  - `url` : définit l'URL de la requête
  - `type` : précise la méthode (`GET` ou `POST`)
  - `data` : précise les données à envoyer si `POST`
  - `success` : définit traitement à effectuer si requête réussit (fonction)
  - `error` : définit traitement à effectuer si requête échoue (fonction)

6.5.1 Exemple

```
//...
$.ajax( {
    url: "test.php",
    error: function(resultat, statut, erreur) {
        // ...
    }
    success: function(resultat, statut) {
        // ...
    }
} );
```

6.6 Exemple d'application

Construction de formulaires dynamiques avec Symfony

Take away

- JavaScript (programmation événementielle)
- manipulation du DOM
- JQuery
  - Sélecteurs similaires à ceux de CSS
- AJAX : requêtes asynchrones

Postface

Crédits illustrations et vidéos

— Logo *JQuery* (source : `https://brand.jquery.org/logos/`)

Copyright