Architecture(s) et application(s) Web

CSC4101 - Interface dynamique côté navigateur en Javascript

23/10/2023

Plan de la séquence

er au-delà des formulaires basiques Sy

Edition des OneToMany

Version de base

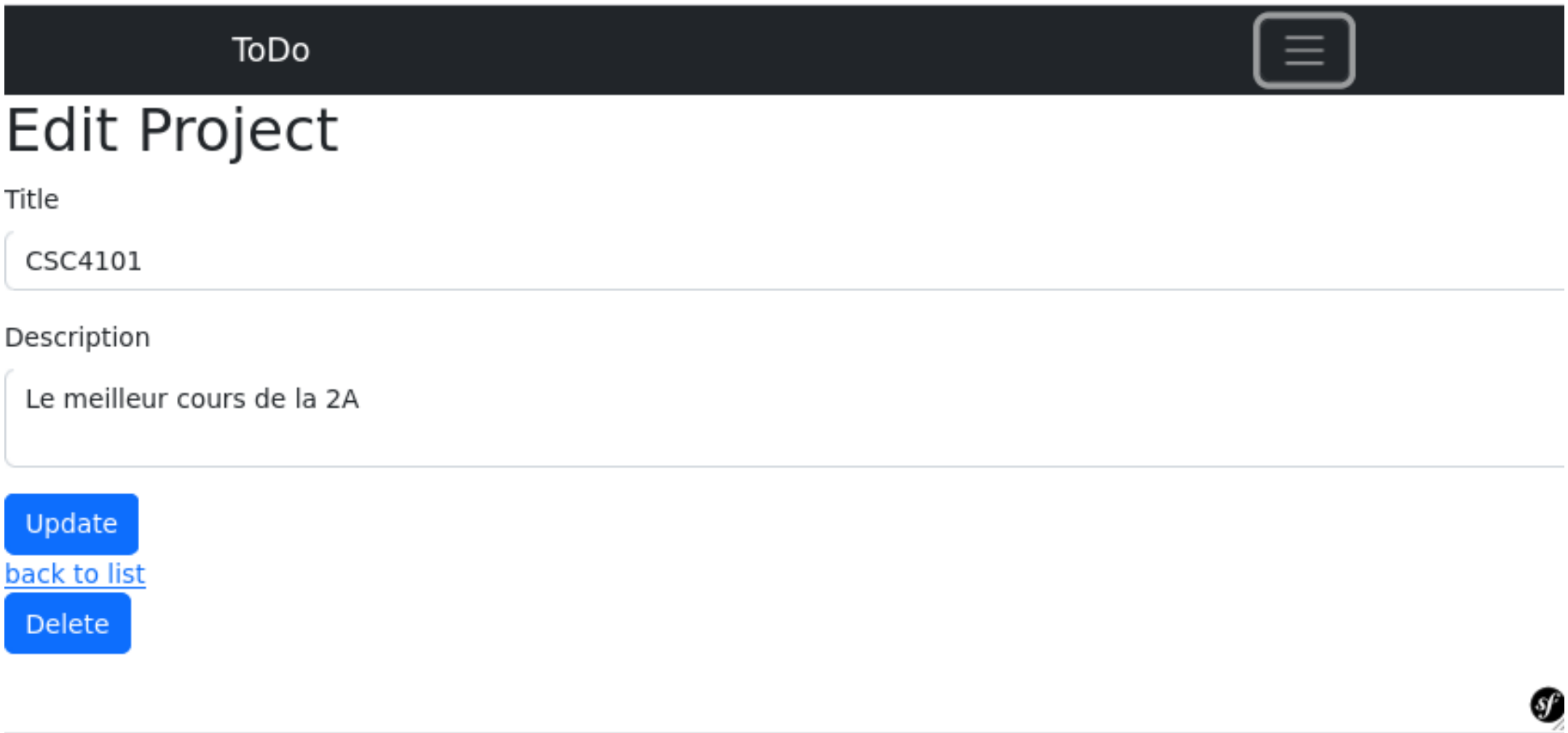


Figure 1 : version du formulaire d'édition généré par `make:crud`

Ajout d'une association OneToMany

```php
<?php

//...

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

Champ par défaut pour propriété multi-valuée

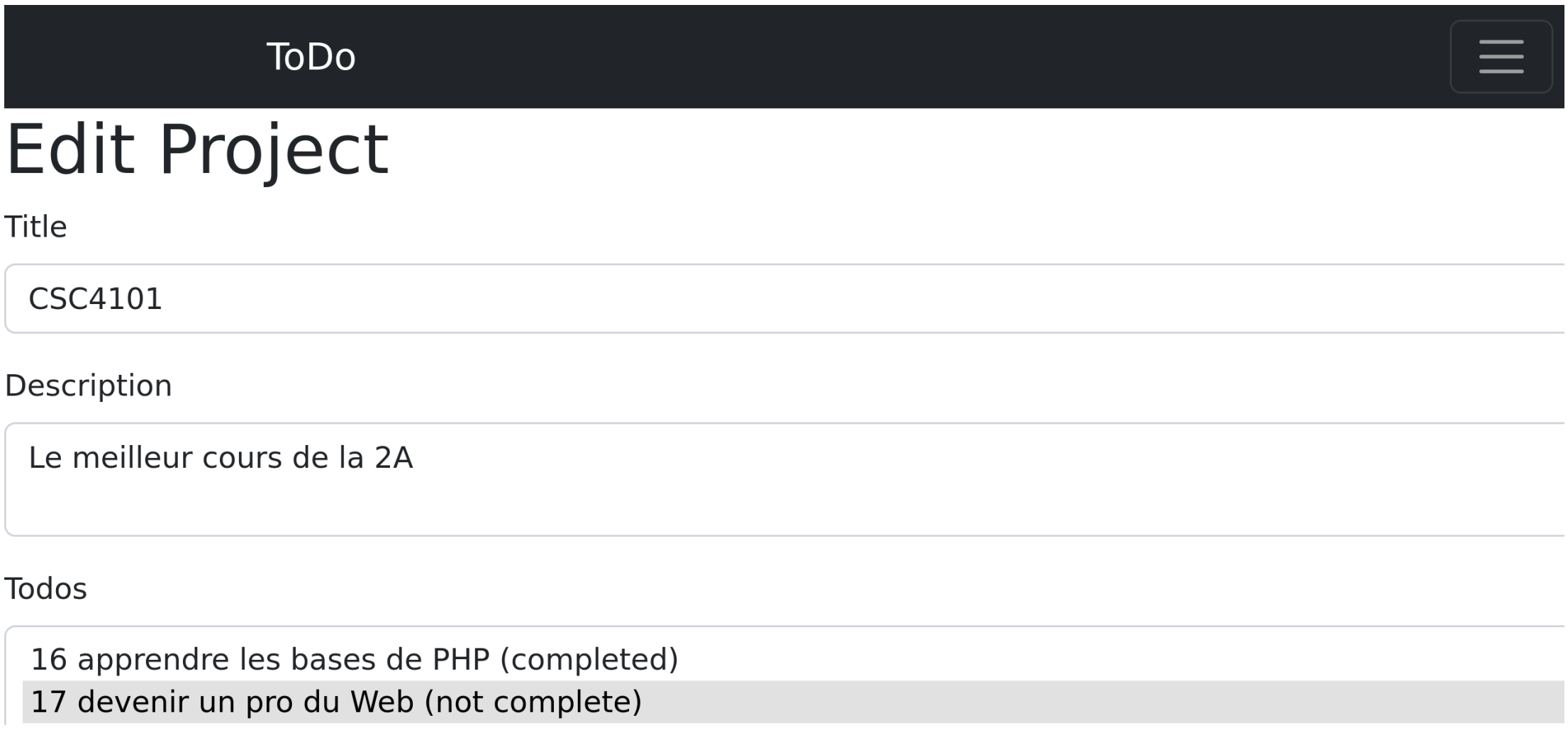


Figure 2 : ajout du champ todos

Problème : comment sélectionner plusieurs valeurs ?

Version à *checkboxes*

```php
<?php

//...

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

Champ par défaut pour propriété multi-valuée

CSC4101

Description

Le meilleur cours de la 2A

Todos

- [x] 16 apprendre les bases de PHP (completed)
- [ ] 17 devenir un pro du Web (not complete)
- [x] 18 monter une startup (not complete)
- [x] 19 devenir maître du monde (not complete)
- [ ] 20 (not complete)
- [x] 21 (not complete)

Update

Figure 3 : sélection multiple via *checkboxes*

Cool ! … mais `__toString()` :-/

Mais... sauvegarde ne marche pas

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

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

Mise en forme du formulaire

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 %}
```

gabarit project/_form.html.twig

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

Affichage en colonnes

ToDo

Edit Project

Title

CSC4101

Description

Le meilleur cours de la 2A

Update

back to list

Todos

- [x] 16 apprendre les bases de PHP (comple
- [x] 17 devenir un pro du Web (not complet
- [ ] 18 monter une startup (not complete)
- [x] 19 devenir maître du monde (not compl
- [ ] 20 (not complete)
- [ ] 21 (not complete)

Figure 4 : séparation des champs en plusieurs colonnes

gabarit `project/_form.html.twig` en colonnes

```
{{ 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) }}
```

Surcharge gabarit par défaut des *checkboxes*

Figure 5 : affichage des tâches en HTML stylé

gabarit `project/edit.html.twig` avec surcharge gabarit checkboxes

```
{% 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 %}
```

Plan de la séquence

erfaces dynamiques

Vues générées côté serveur

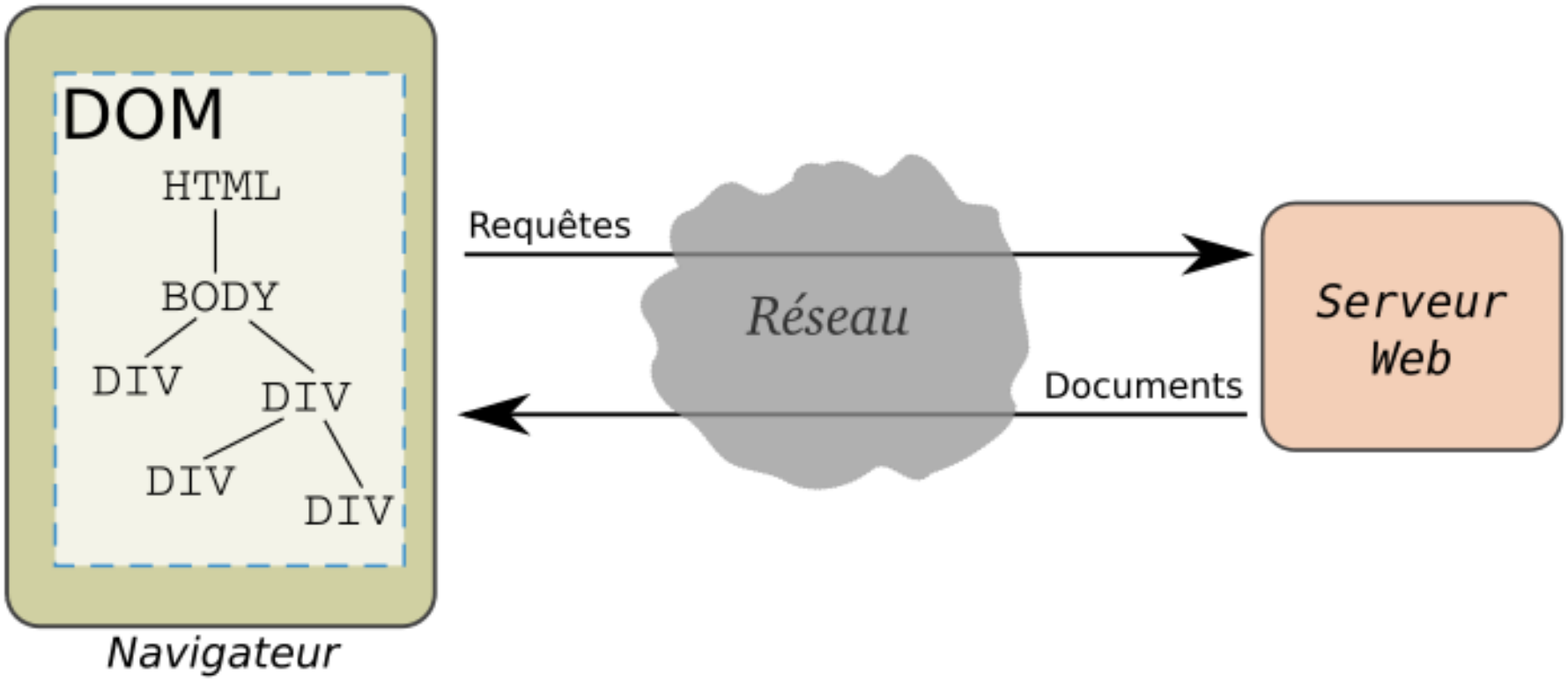


Figure 6 : Documents statiques

Vues générées côté client

« HTML dynamique »

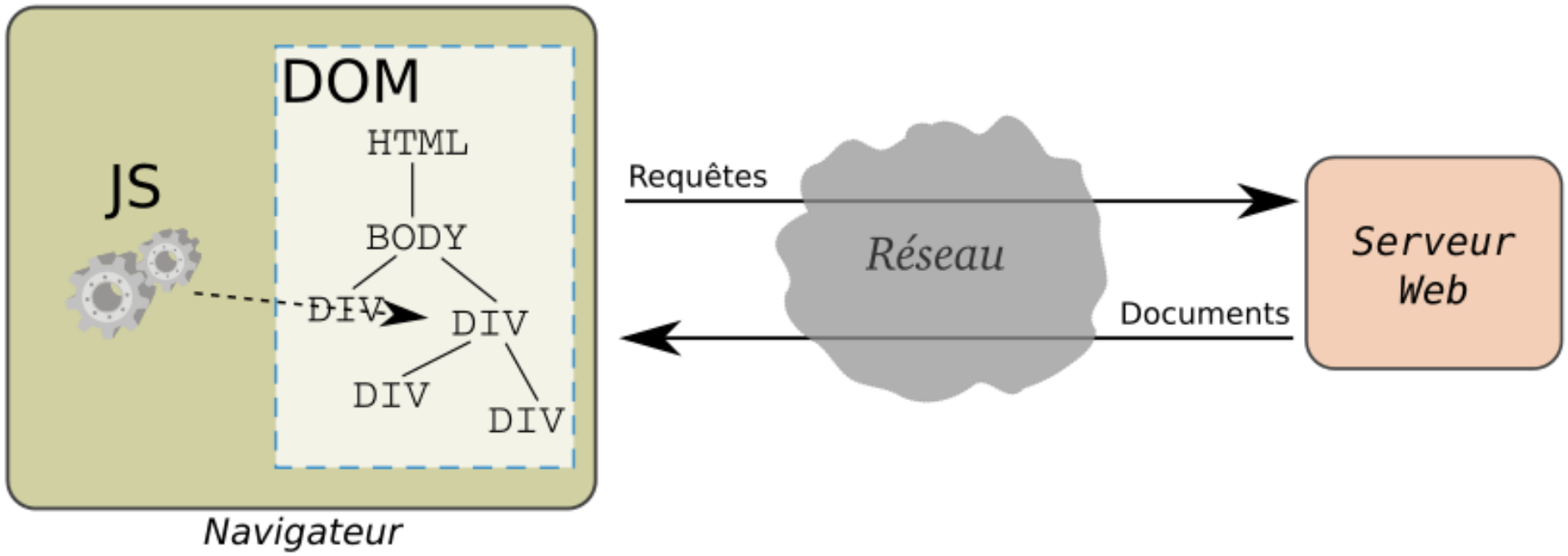


Figure 7 : Documents dynamiques avec Javascript

</principes>

Plan de la séquence

2. JavaScript

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*)

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

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)

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*)...
- Gestion des **événements**
  - Capter les évènements : clics, déplacement de la souris, chargement de la page ...
  - Associer des fonctions aux événements

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"
}
```

Utilisation

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>
```

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");
}
```

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`...)

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);">
```

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

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>
```

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

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
```

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.

Objets prédéfinis 2/2

Les objets du navigateur :

- fenêtre,
- document,
- formulaires
- ...

Hiérarchie 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.

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

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

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)*

</JavaScript>

Plan de la séquence

3. Manipulation du DOM

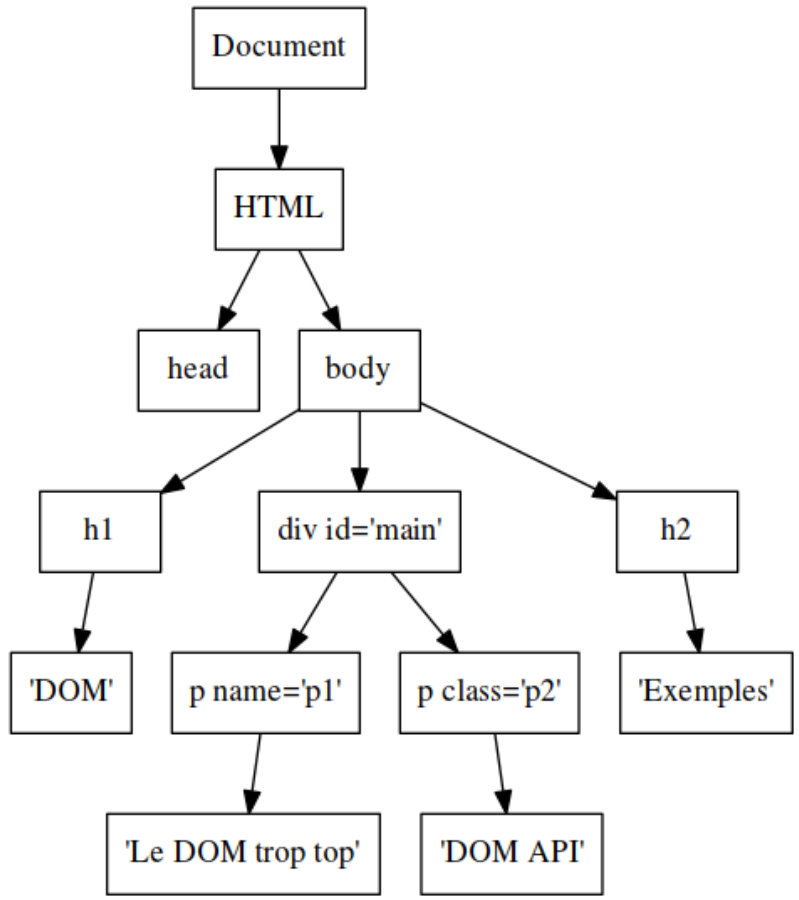
DOM

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)

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>
```



Manipulation

- 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.

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`, …

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

</manipulation_DOM>

Plan de la séquence

4. *Framework* JavaScript *JQuery*

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_JavaSc based_web_frameworks

jQuery

https://jquery.com/

- Bibliothèque des plus populaires et des plus utilisées
- Utilisable avec tous les navigateurs pour un résultat identique (avant la standardisation)

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

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*

Chargement de JQuery

- télécharger depuis jQuery.com et inclure en local

```
<head>
  <script src="jquery-3.6.0.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-3.6.0.min.js">
  </script>
</head>
```

Attention : traçage des utilisateurs

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).

Structure générale des programmes

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

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

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
    //...
});
```

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...

});
```

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

</JQuery>

Plan de la séquence

5. AJAX

AJAX ?

Figure 8 : *Illustration When you say AJAX*

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.

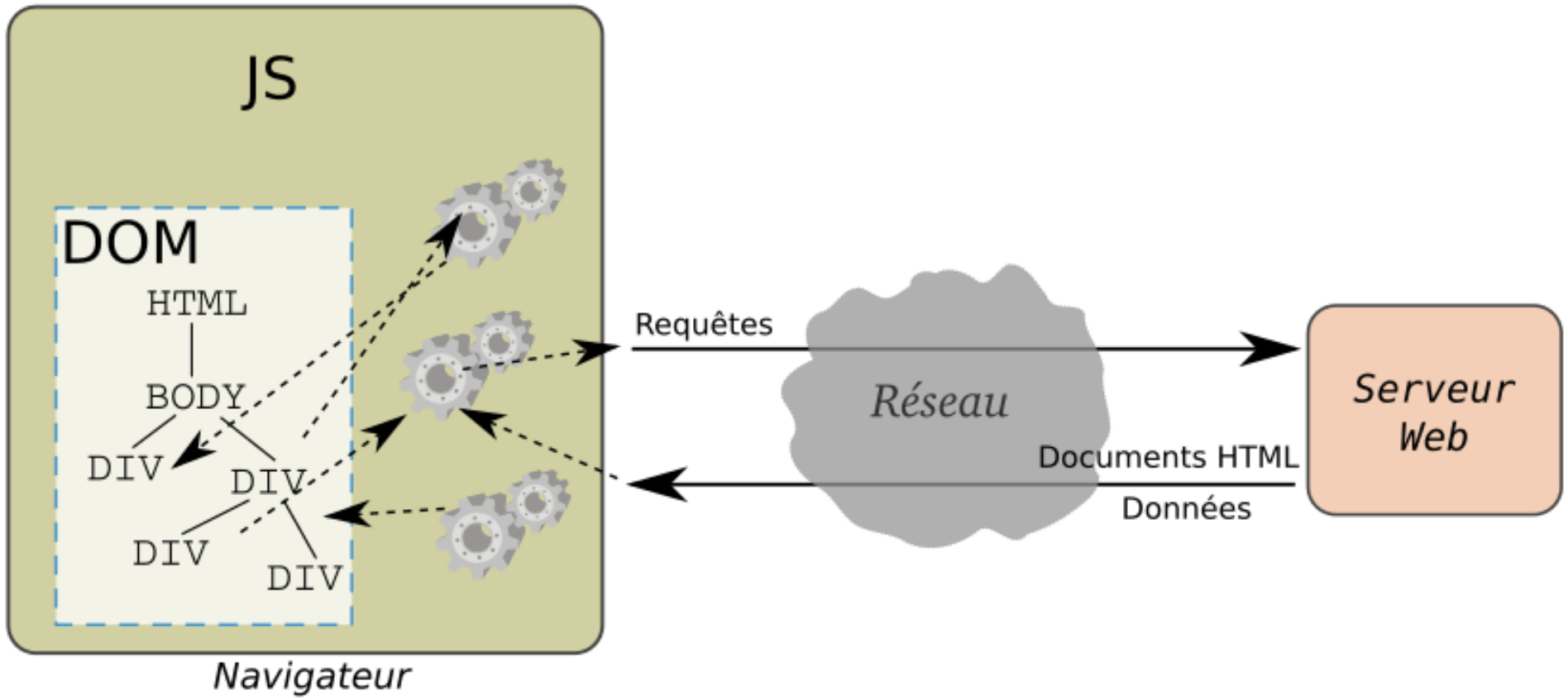
Classe `XMLHttpRequest` (XHR) de JavaScript

- 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)

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…)

Sous-requêtes HTTP



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)

Exemple

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

Exemple d’application

Construction de formulaires dynamiques avec Symfony

</AJAX>

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/)
- Illustration « *When you say AJAX* » (auteur inconnu) - source : https://9gag.com/gag/6304578/when-you-say-ajax

Copyright

- Document propriété de ses auteurs et de Télécom SudParis (sauf exceptions explicitement mentionnées).
- Réservé à l'utilisation pour la formation initiale à Télécom SudParis.