Secteur Tertiaire Informatique
Filière « Etude et développement »

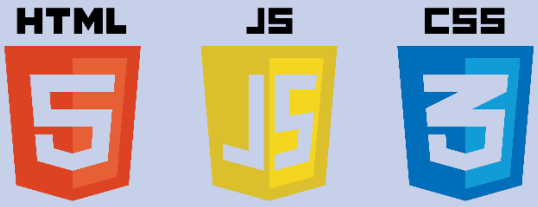
Séquence « Développer des pages Web »

Support 3 JavaScript – JQuery

Apprentissage

Support

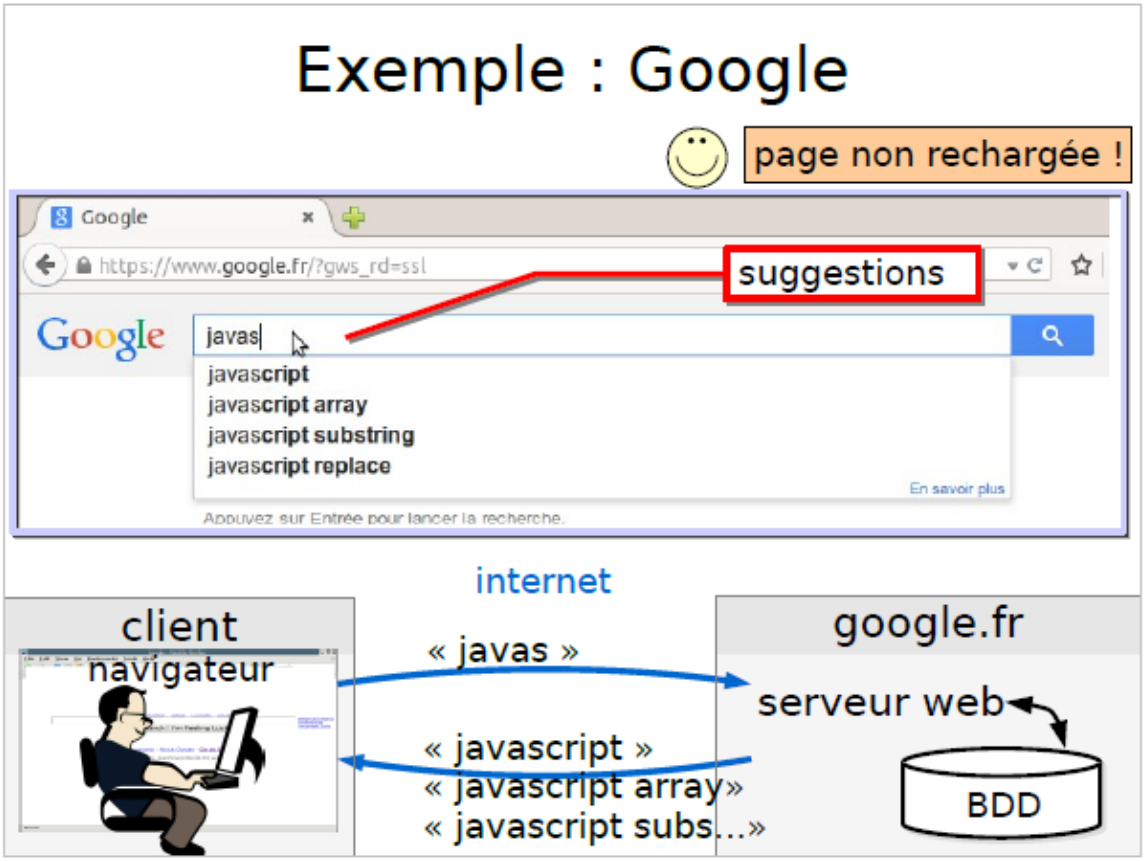
Evaluation



Afpa © 2019 – Section Tertiaire Informatique – Filière « Etude et développement »

TABLE DES MATIERES

1. AJAX



Quand un utilisateur commence à taper une recherche dans Google, Google lui propose une **liste de suggestions**.
Il est impossible d'inclure dans la page d'origine toutes les suggestions possibles.
Le navigateur va donc demander au serveur la liste de suggestions au fur et à mesure que l'utilisateur écrit. Ceci se fait **sans recharger la page**.

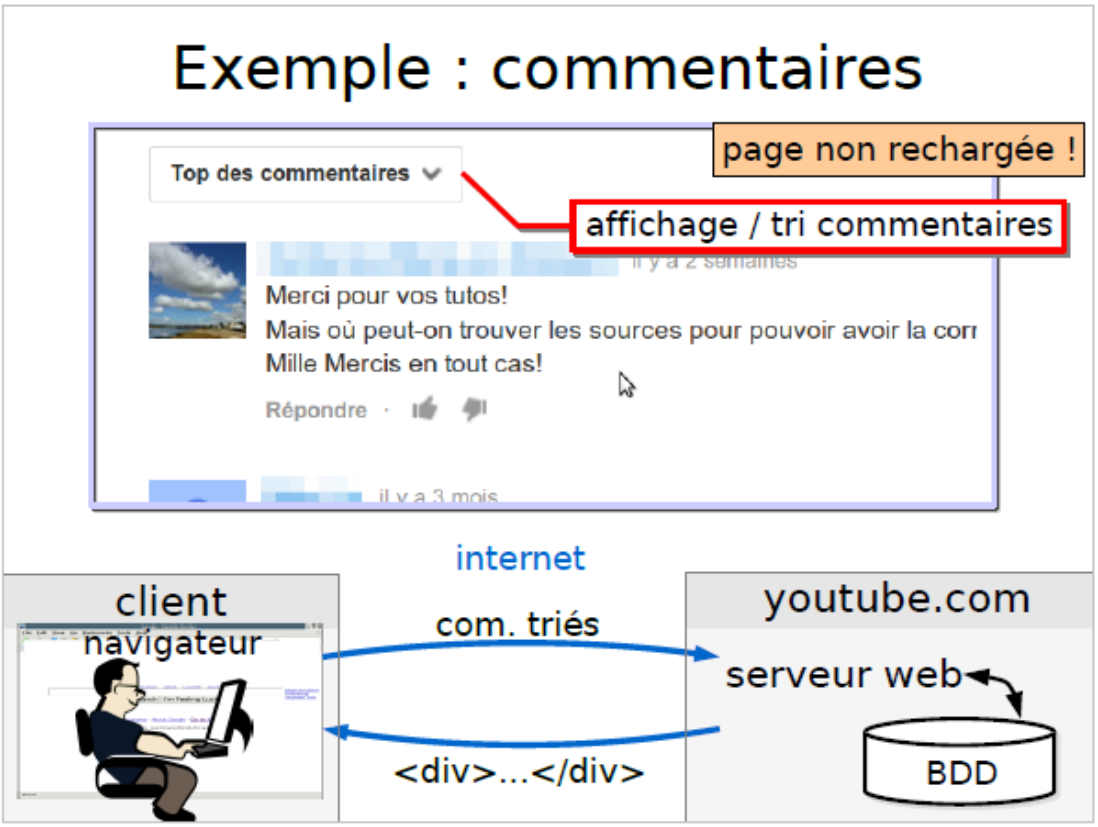
Dans ce qu'on a vu jusqu'à maintenant, toutes les interactions avec le serveur se faisaient au moment du chargement de la page.

Ici, la liste des suggestions est cherchée à partir du serveur, à partir du JS et **après la fin du chargement de la page**.

Dans ce deuxième exemple, l'utilisateur appuie sur un bouton « **J'aime** » dans les commentaires d'une vidéo sur **Youtube**.
Son action est enregistrée sur le serveur, mais **la page ne se recharge pas**.

Le fonctionnement habituel en HTML du bouton consiste à envoyer une requête POST et afficher une nouvelle page.
Le rechargement de la page est désagréable pour l'utilisateur. C'est lent, la vidéo s'interrompt, il perd sa position dans la page...

Ici, la **requête sur le serveur** se fait **sans rechargement de la page**.



Sur la même page **Youtube**, l'**affichage des commentaires** se fait après la fin de de l'affichage de la page. A la place des commentaires, un message indique pendant un bref instant « **chargement en cours...** ».

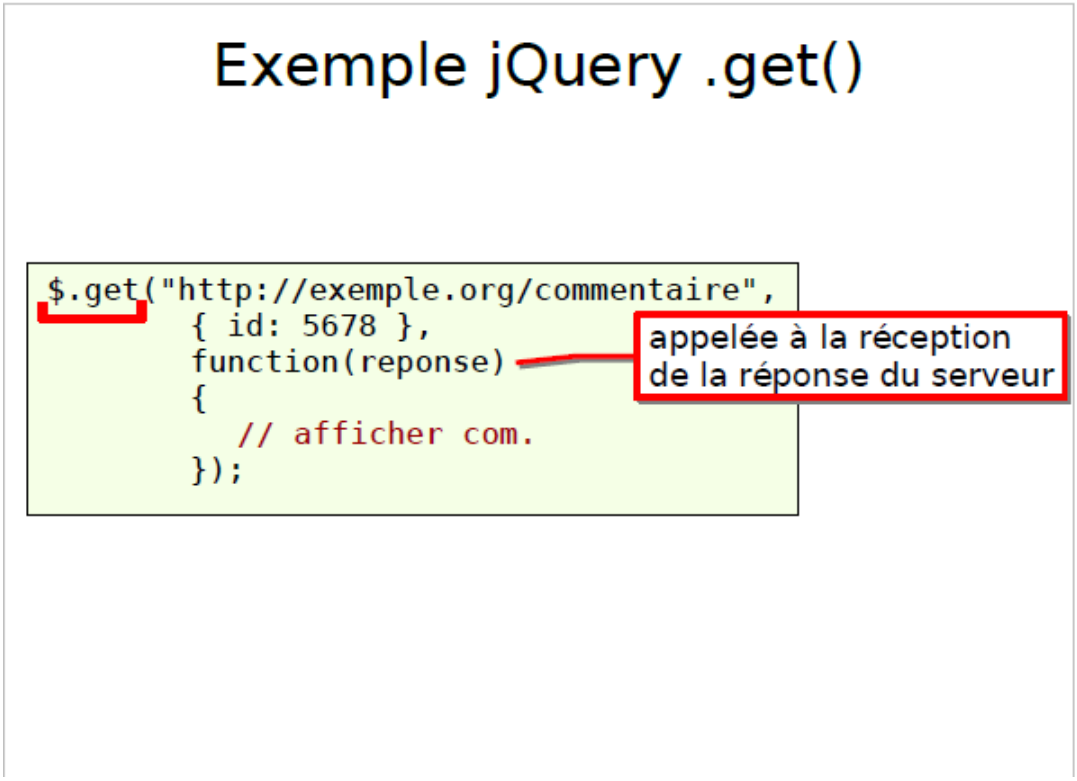
Par ailleurs, l'utilisateur peut **trier les commentaires**, **sans recharger la page**. Une vidéo peut avoir des milliers de commentaires.

Ils ne peuvent être chargés qu'à la demande.

AJAX est une approche dans laquelle on **communique** avec le **serveur**, à partir du **JavaScript**, **sans recharger la page**.

La requête au serveur peut prendre du temps. Il serait très désagréable que le navigateur soit bloqué pendant cette attente. La requête est donc faite de manière "**asynchrone**" : on lance la requête en JS et on fournit **une fonction qui sera appelée plus tard**, lorsque que la requête sera finie.

Le "X" d'AJAX vient de "XML". Historiquement la communication entre le serveur et le client se faisait en XML ... aujourd'hui on préfère un autre format appelé **JSON**.

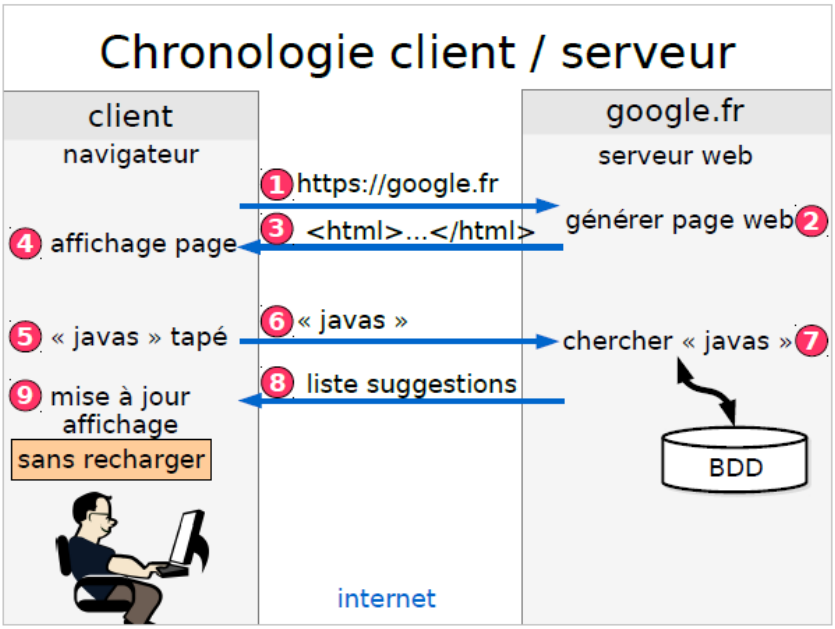


La fonction **jQuery .get()** permet de faire une **requête AJAX** en utilisant la méthode "GET".
Remarquez la syntaxe « **$.get** » : c'est une fonction qui est appelée directement sur l'objet jQuery ($) et pas sur une liste d'éléments jQuery.

Ici, **$.get** prend **3 arguments** :

1) l'URL vers lequel on fait la requête
2) des données GET à envoyer au serveur
3) la fonction qui sera appelée quand la requête aura réussi

Important: cette **fonction** (3) n'est **pas appelée tout de suite**, lors de l'appel de $.get ... mais **plus tard**, en **cas de succès de la requête**.



Reprenons l'exemple des suggestions dans la recherche Google. Il est très important de bien comprendre la chronologie et de savoir ce qui se passe sur le client et sur le serveur.
En pratique, les échanges peuvent devenir compliqués.

1) Client au serveur « bonjour google.fr, je veux la page https://google.fr »

2) Le **serveur** de Google **exécute un programme** qui génère la page HTML

3) Serveur au Client : voici la page au format HTML

4) Le Client affiche la page reçue (simplifié, en pratique il y a de nombreuses autres requêtes pour le CSS, le JS, les images...)

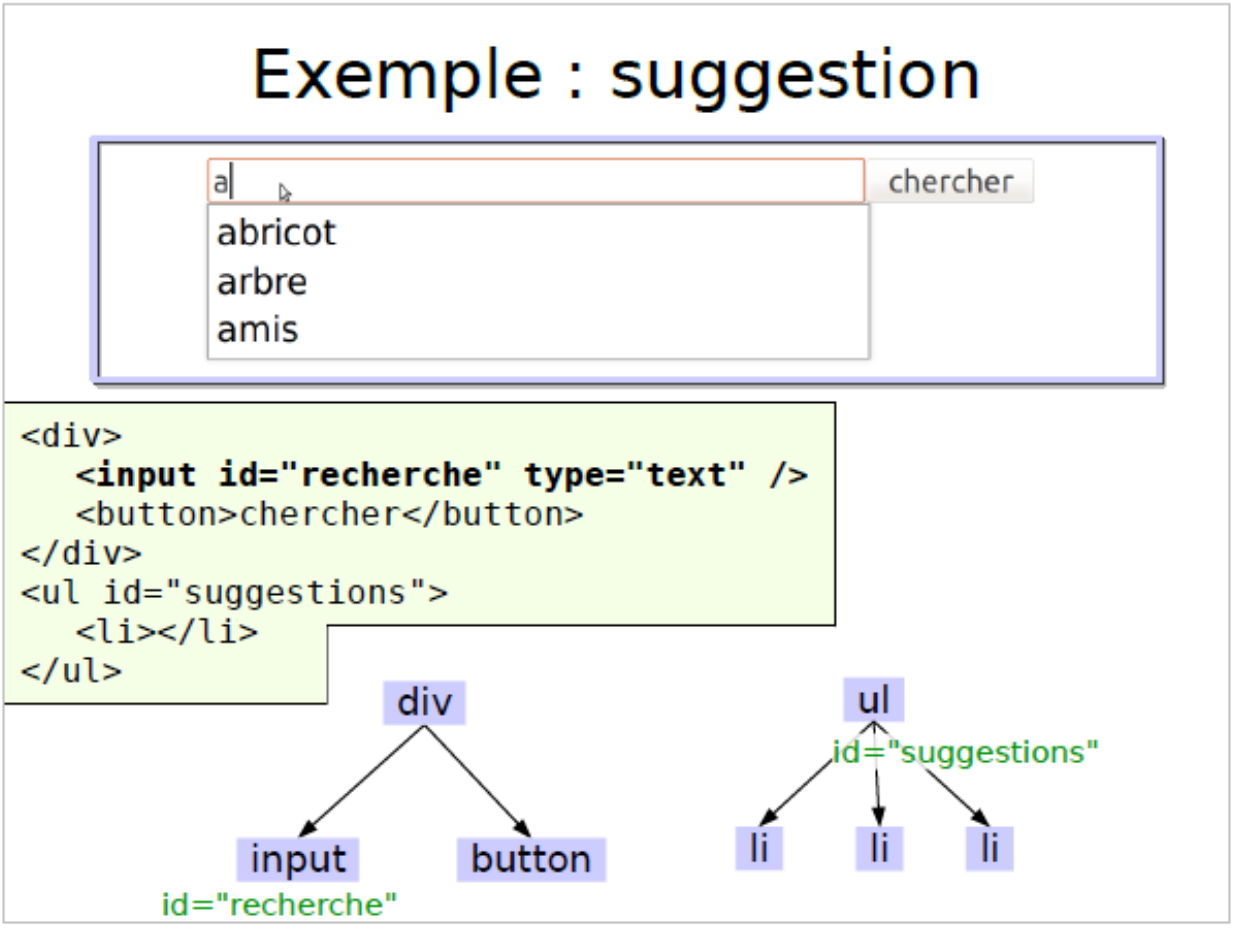
5) L'utilisateur commence à taper « javas ». Un programme JavaScript sur le client réagit à l'événement clavier et **lance une requête AJAX au serveur**.

6) Requête AJAX au serveur

7) Le Serveur reçoit la requête et exécute un programme (par exemple PHP) qui va chercher des suggestions commençant par « javas » dans une Base de Données (ex: sql).

8) Le Serveur envoie les suggestions au client

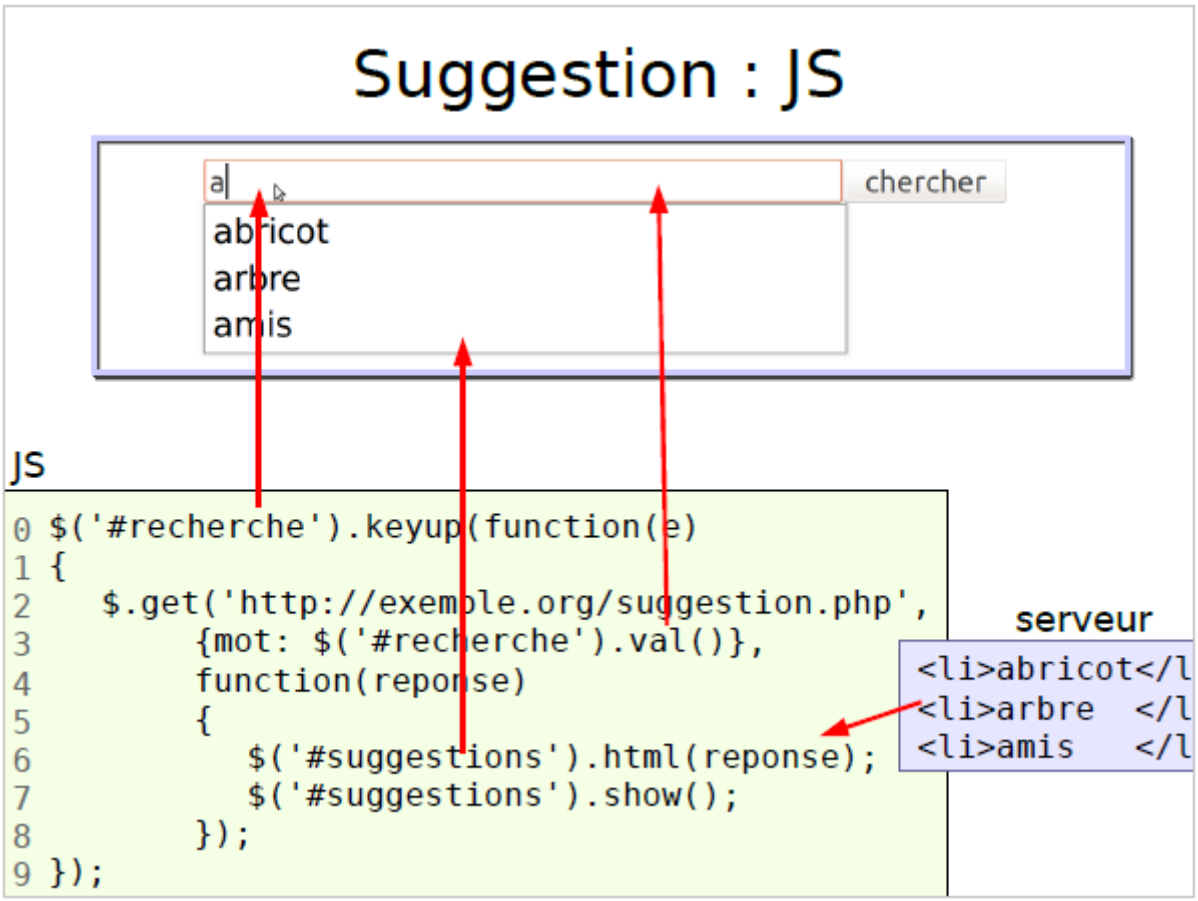
9) Le JavaScript sur le Client affiche les suggestions



On va voir un exemple similaire aux suggestions Google, très simplifié.

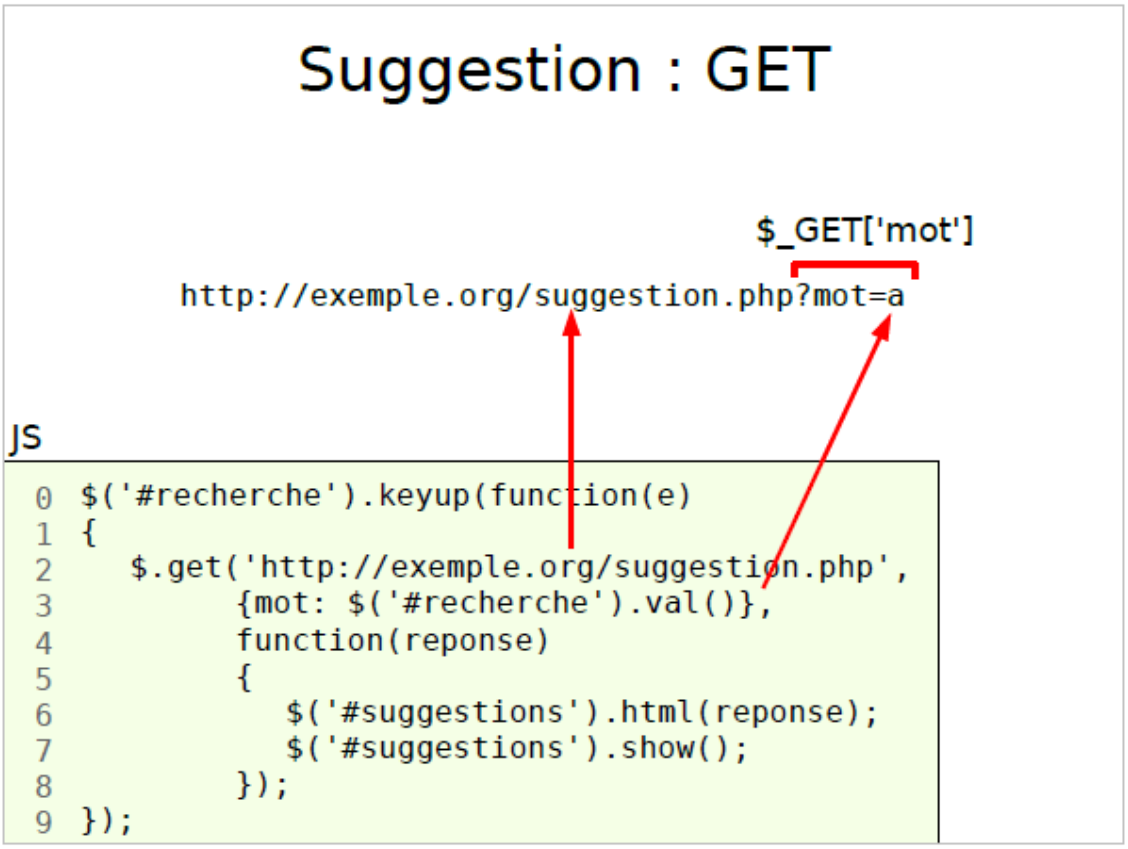
Le **HTML** est très simple: un champ texte **<input>** pour que l'utilisateur puisse taper sa recherche.

En dessous, une **liste <ul> / ** pour afficher les suggestions.



Le début du JS est classique :
$('#recherche') est une **liste jQuery** contenant uniquement le champ texte. On écoute les événements du clavier dans ce champs texte grâce à « **.keyup()** ».
Quand l'utilisateur relâche une touche (keyup), la fonction **$.get** est appelée.
Sont fournis en argument : L'URL, les données pour le serveur et la fonction à appeler après la requête.
Quand la requête réussi, la fonction (ligne 4) est appelée avec en argument (**reponse**) le HTML renvoyé par le serveur.
Ce HTML est affiché dans **<ul id="suggestions">**
Comme la liste est peut être cachée, il faut l'afficher.

Il manque dans cet exemple le code pour remplir le champ texte quand on clique sur un élément de la liste.

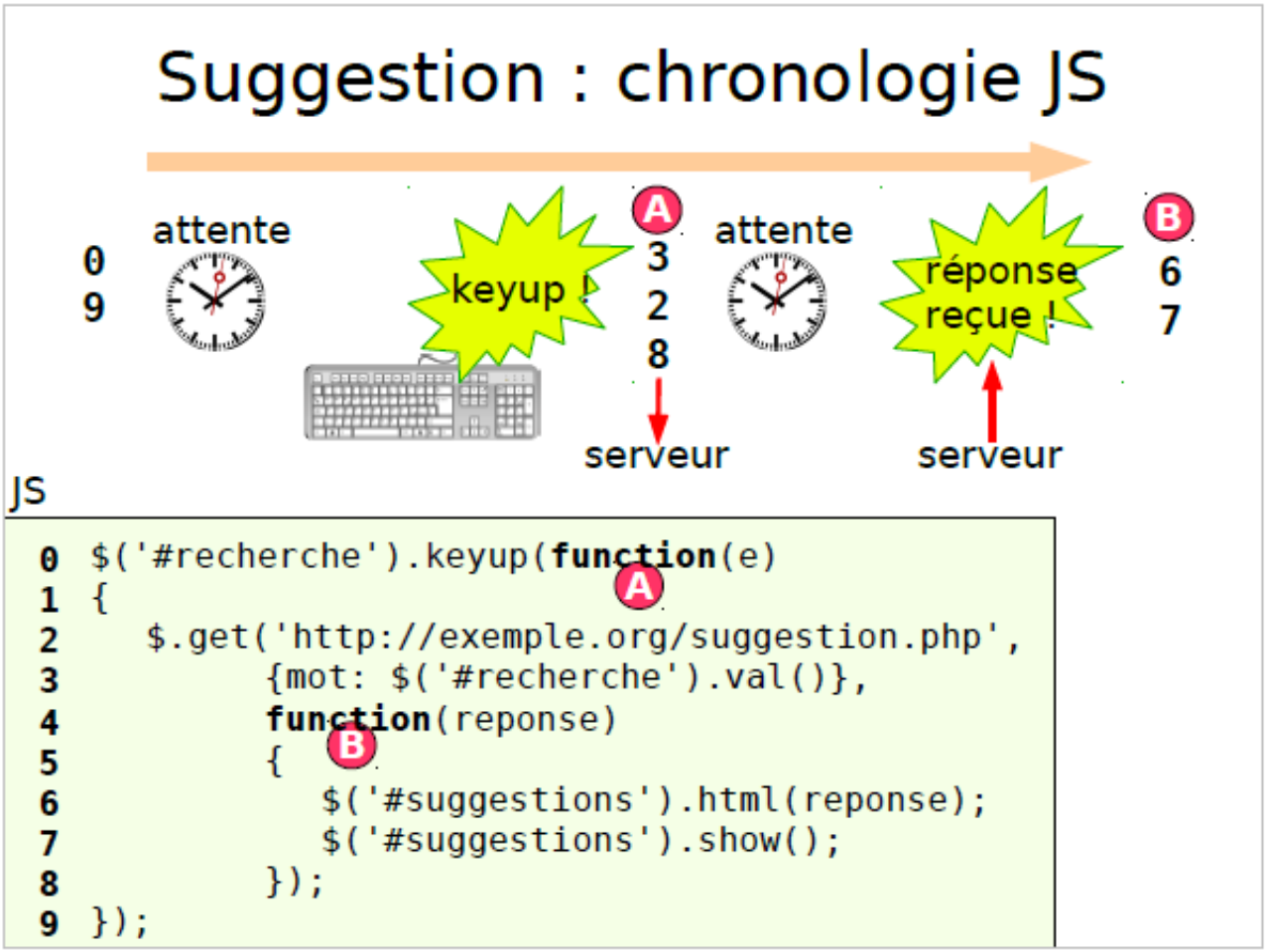


Le 2e argument indique les données fournies au serveur avec la **méthode GET**.

Dans la **méthode GET**, les données sont **inclues dans l'URL** en utilisant le séparateur "**?**" puis ensuite les séparateurs "**&**".

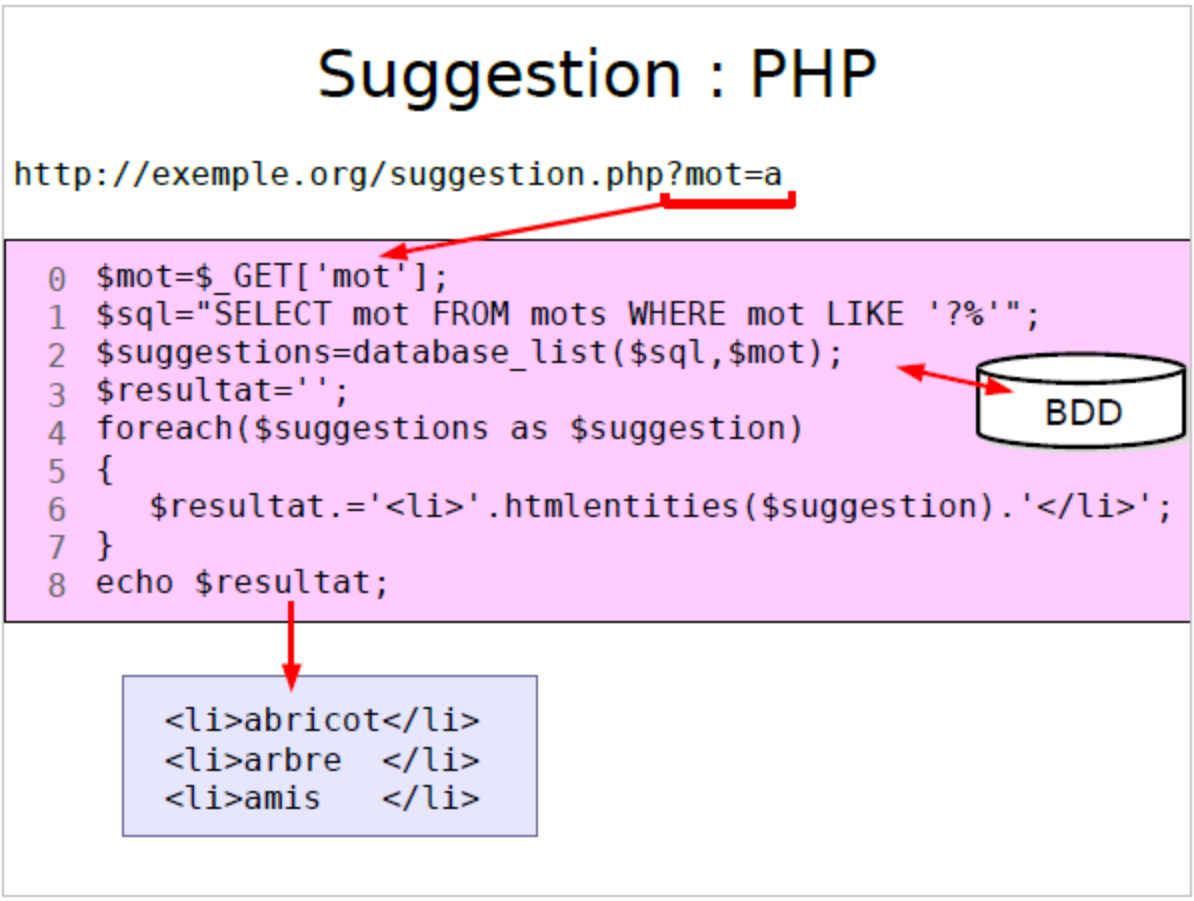
Remarque: on aurait pu mettre ces informations directement dans l'URL de l'argument 1... mais il faudrait les encoder correctement (espaces et autres caractères spéciaux).

JQuery le gère plus simplement.



En JS, les **fonctions anonymes** permettent d'écrire très simplement du **code événementiel** et **différé**.

Cette simplicité peut nous faire oublier la chronologie compliquée de l'exécution de ce programme.

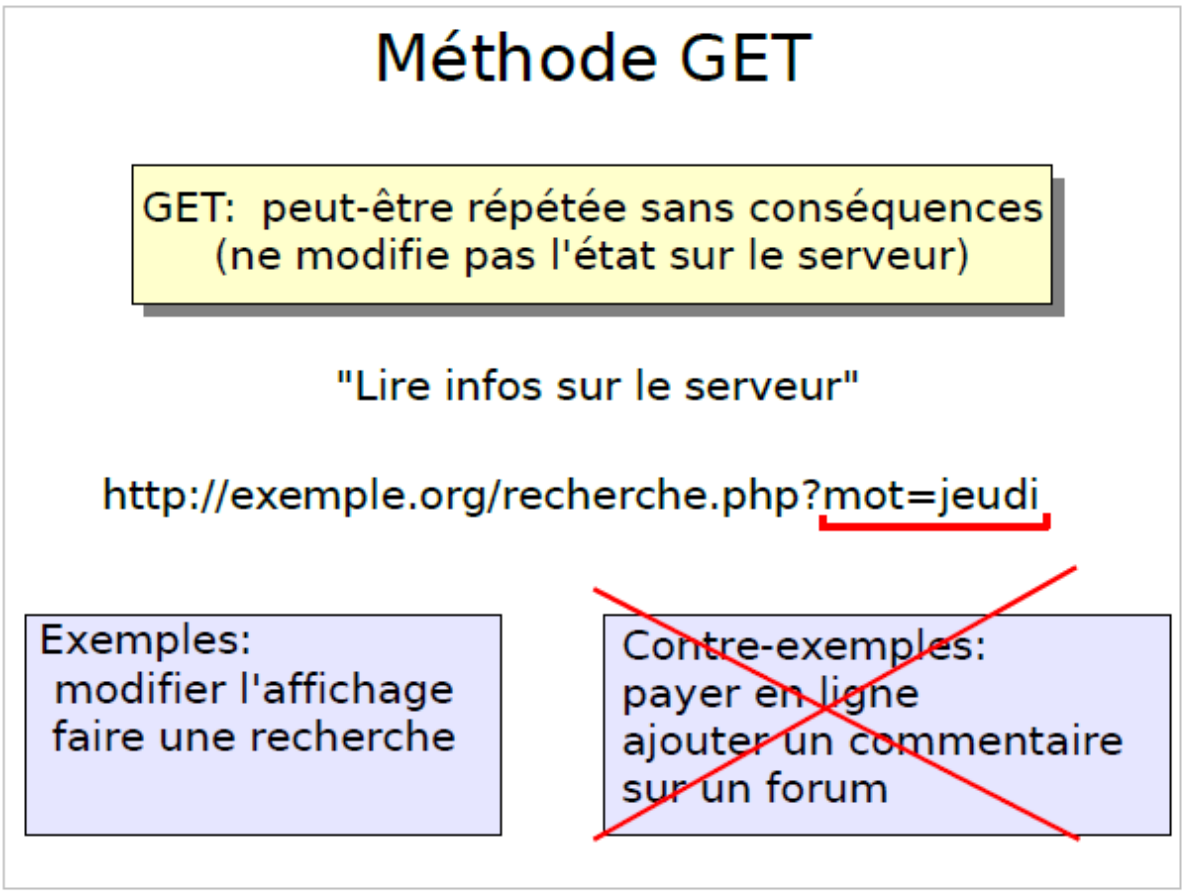


En PHP, les **arguments GET** sont récupérés dans le **tableau global $_GET**.

Dans cet exemple, le programme fait une requête SQL à la base de données pour chercher tous les mots qui commencent pas "a".

On construit alors le HTML contenant les suggestions (**
...
**...).

Le HTML est envoyé au client tout simplement avec "**echo**"...

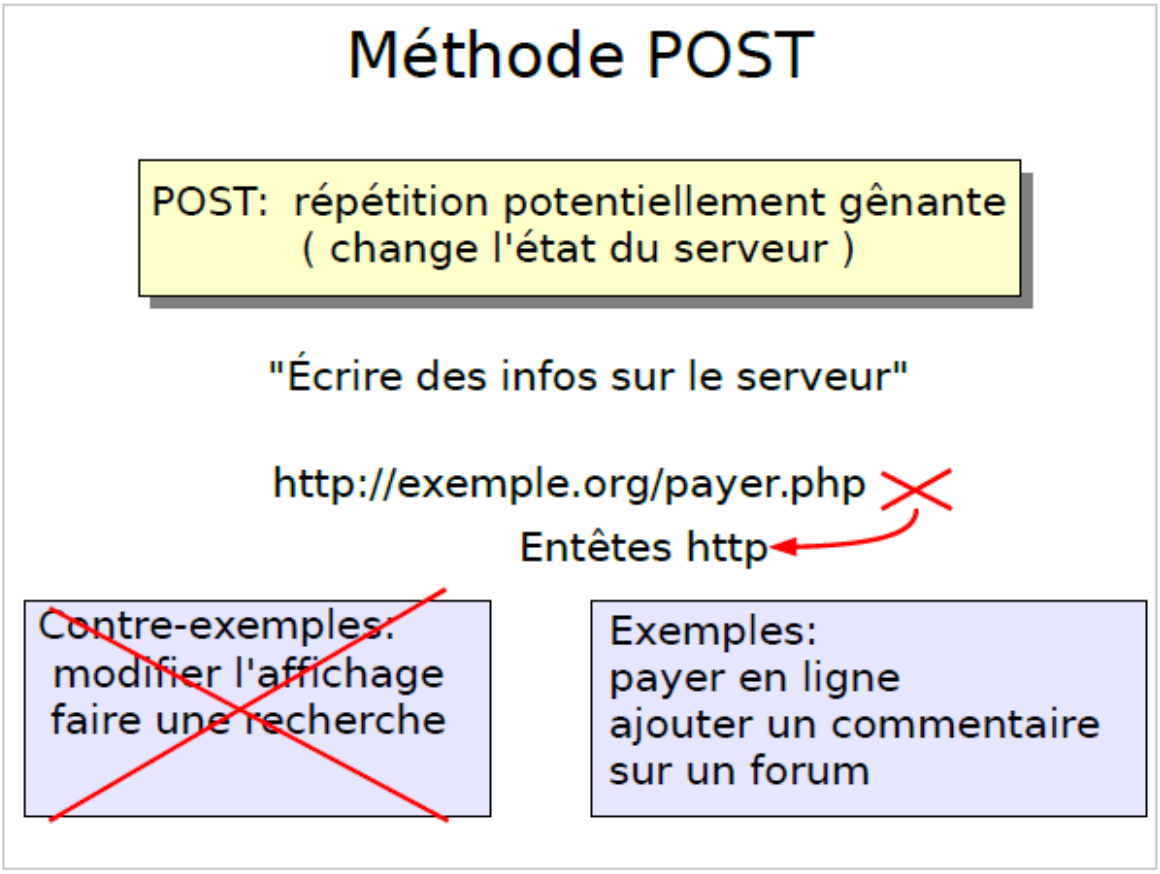


Chaque fois qu'on écrit une **requête AJAX**, comme chaque fois qu'on écrit un formulaire, il faut choisir entre la méthode **GET** et la méthode **POST**.

La principale motivation de ce choix est définie par les normes : les **requêtes GET** sont "**idempotentes**", elles **peuvent être répétées** sans conséquences.

Les données des requêtes GET sont transmises dans l'URL.

Ceci a conduit de nombreuses personnes à penser, à tort, que le choix entre GET et POST était une question de sécurité.



Les **requêtes POST** sont choisies pour des actions **qui ne peuvent pas être répétées** sans conséquences.

Les **données** sont **transmises** dans les **entêtes de la requête** et **pas dans l'URL.**
Elles ne sont pas visibles dans l'URL, mais elles ne sont pas pour autant "cryptés".

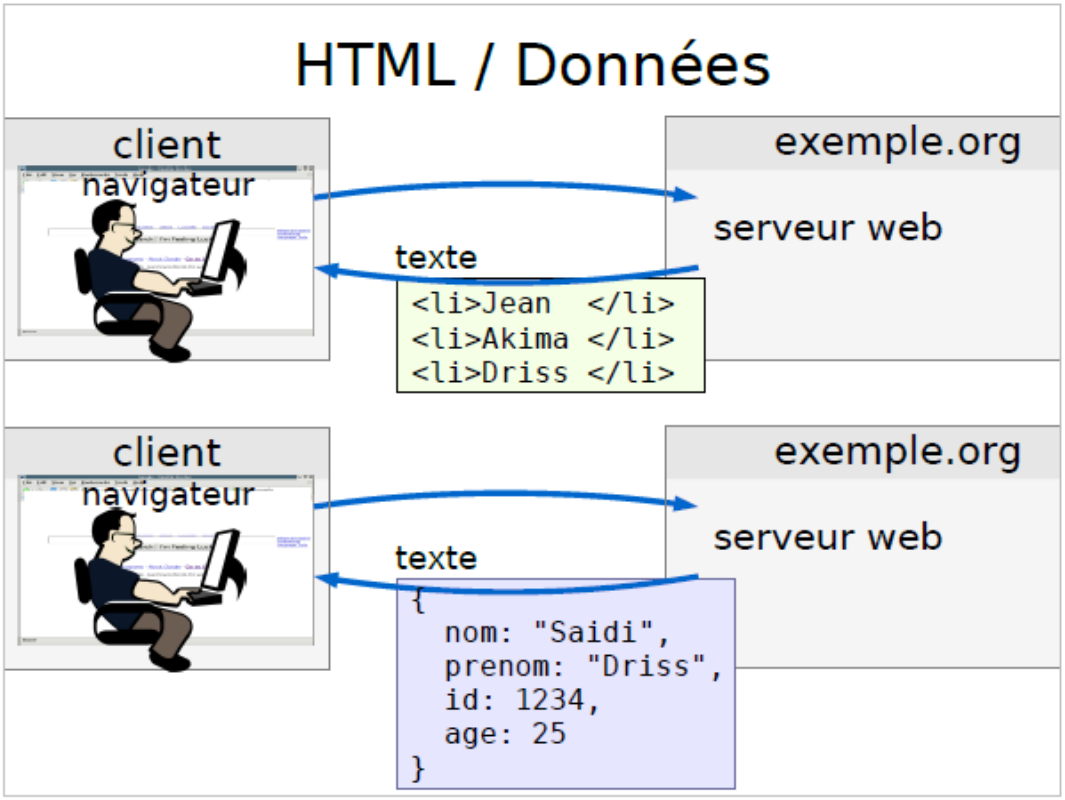
L'argument de sécurité est souvent mal compris.

Dans cet exemple, l'utilisateur clique sur le bouton "J'aime" en dessous d'un commentaire.

Cette action **va modifier l'état du serveur** (+1 à la valeur j'aime sur ce commentaire dans la base de données). Il faut donc utiliser **POST**.

$.post() s'utilise de la même manière que **$.get()**

2. JSON



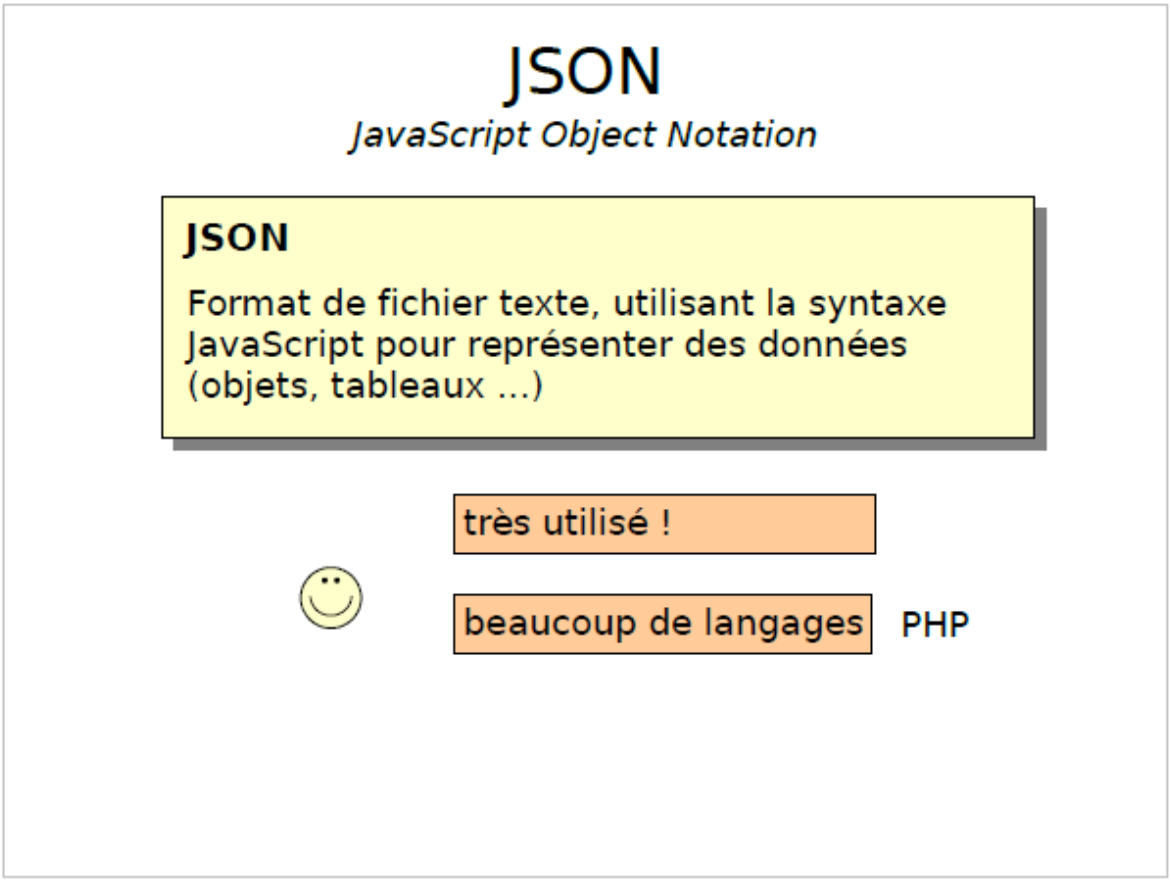
Dans les exemples vus précédemment, le serveur renvoyait un fragment de HTML qui était ensuite affiché.

Bien souvent, on veut chercher des données complexes sur le serveur, et pas juste du HTML.

On pourrait utiliser n'importe quel format pour ces données. Il suffit juste que le programme sur le serveur et le JavaScript soient d'accord.

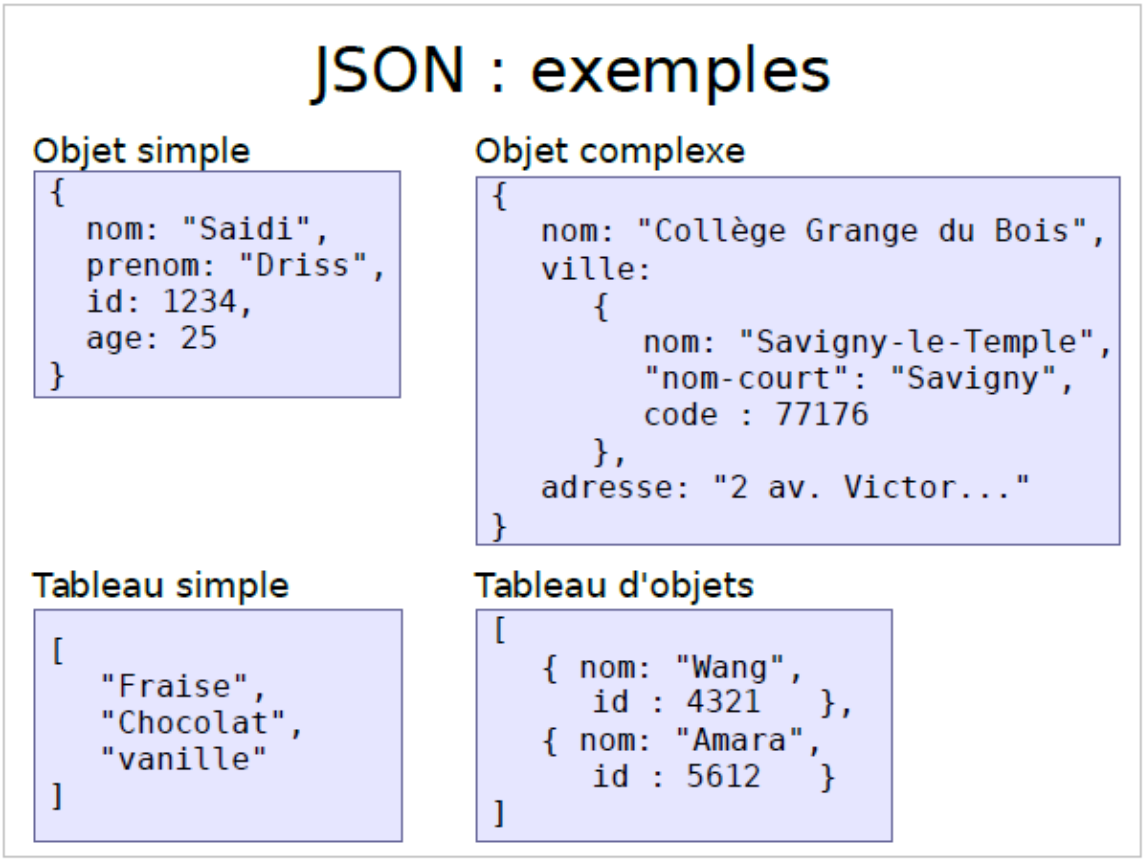
Le XML a été beaucoup utilisé, mais est lourd à manipuler.
Pour les tâches simples, du texte brut est possible.

Pour les informations plus structurées on utilise souvent, aujourd'hui, un **format** appelé **JSON**.



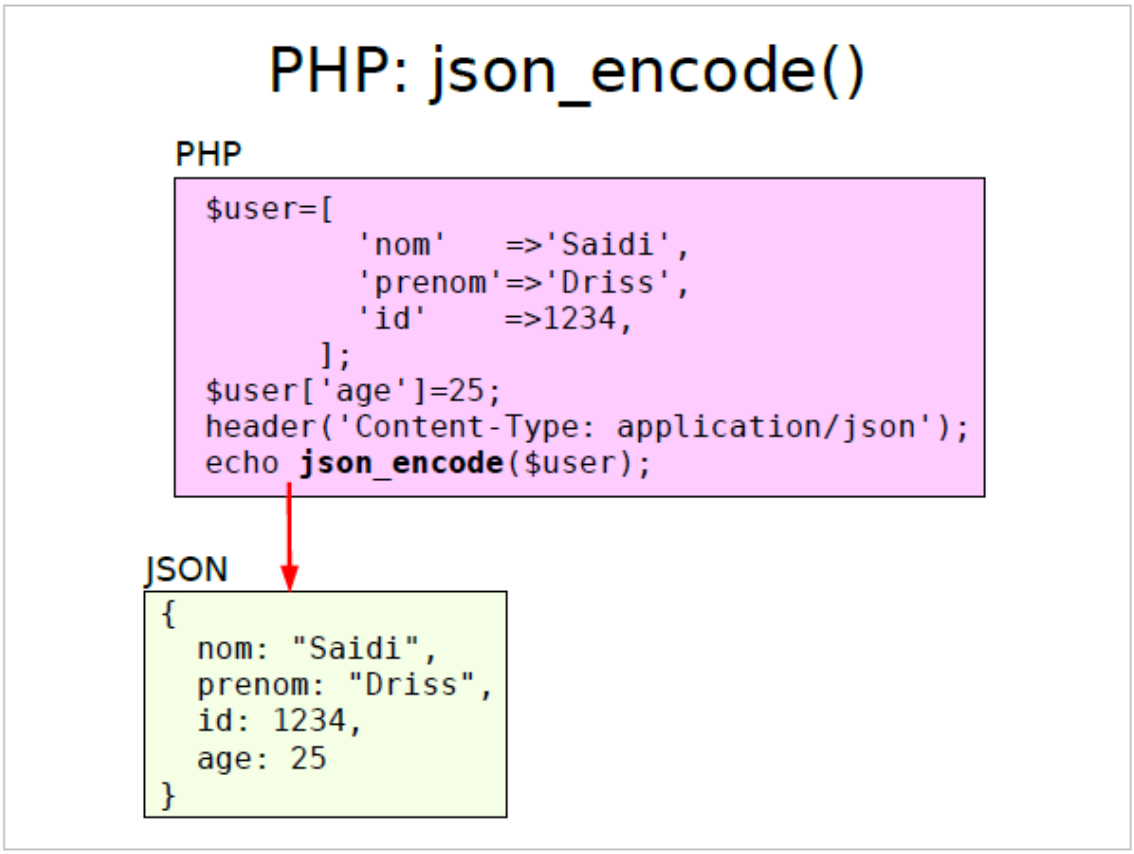
Le **JSON** est au **format texte**, il est presqu'identique au format utilisé en JavaScript pour déclarer des Objets ou des Tableaux.

Tous les principaux langages fournissent des méthodes pour **encoder** et **décoder** le **JSON**.



Voici quelques exemples.

En pratique, les **données JSON** peuvent être **volumineuses** et **complexes.**



En **PHP** la fonction **json_encode()** permet de transformer très simplement un tableau (ou un objet) en **JSON**.
Ce JSON est ensuite envoyé au client, qui pourra le décoder simplement en un objet (ou tableau) JavaScript.

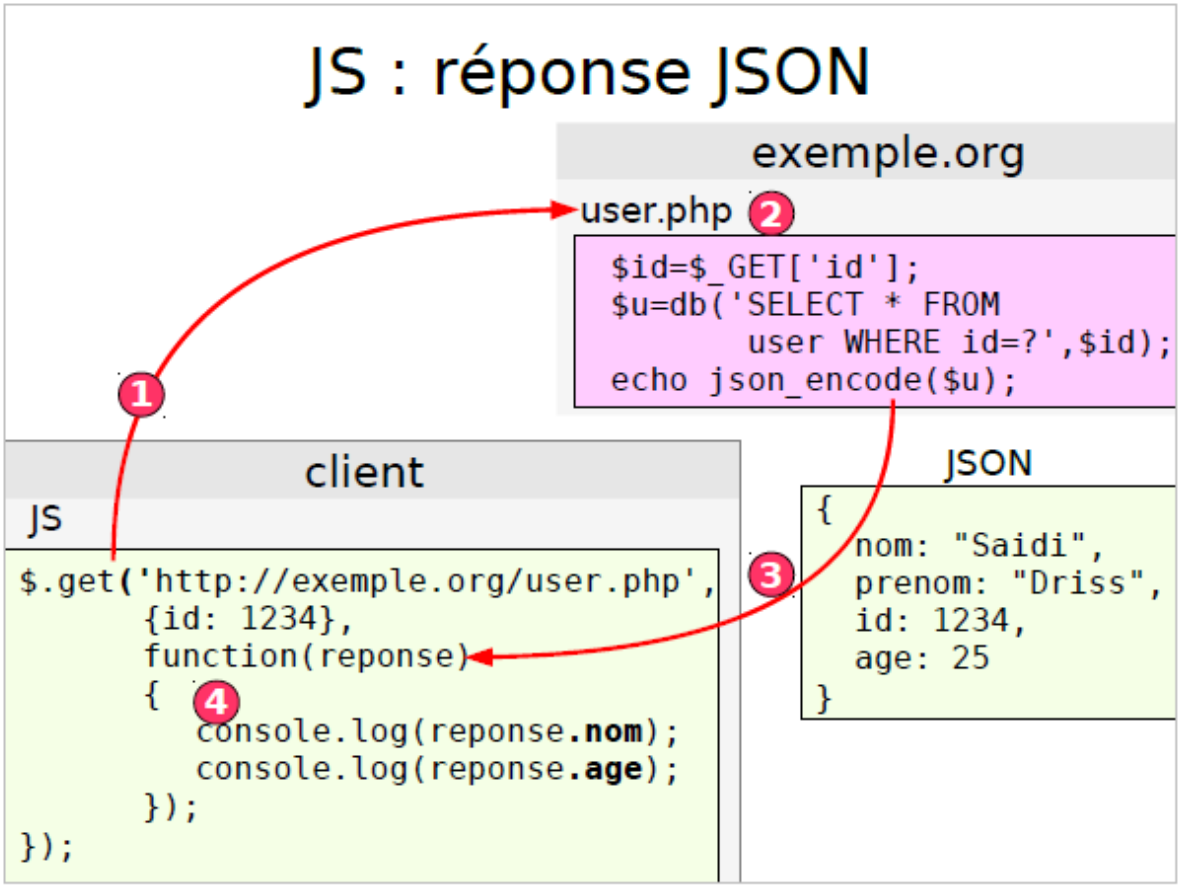
Remarquez la fonction "**header()**". Toute réponse HTTP contient des données appelées "**entêtes**".

Attention: il ne s'agit pas des entêtes du HTML, mais des informations, cachées, en dehors de tout code HTML.

Dans ces entêtes HTTP, on doit préciser le type de contenu transmis:

Pour du HTML: **Content-type**: **text/html**

Pour du JSON : **Content-type**: **application/json**



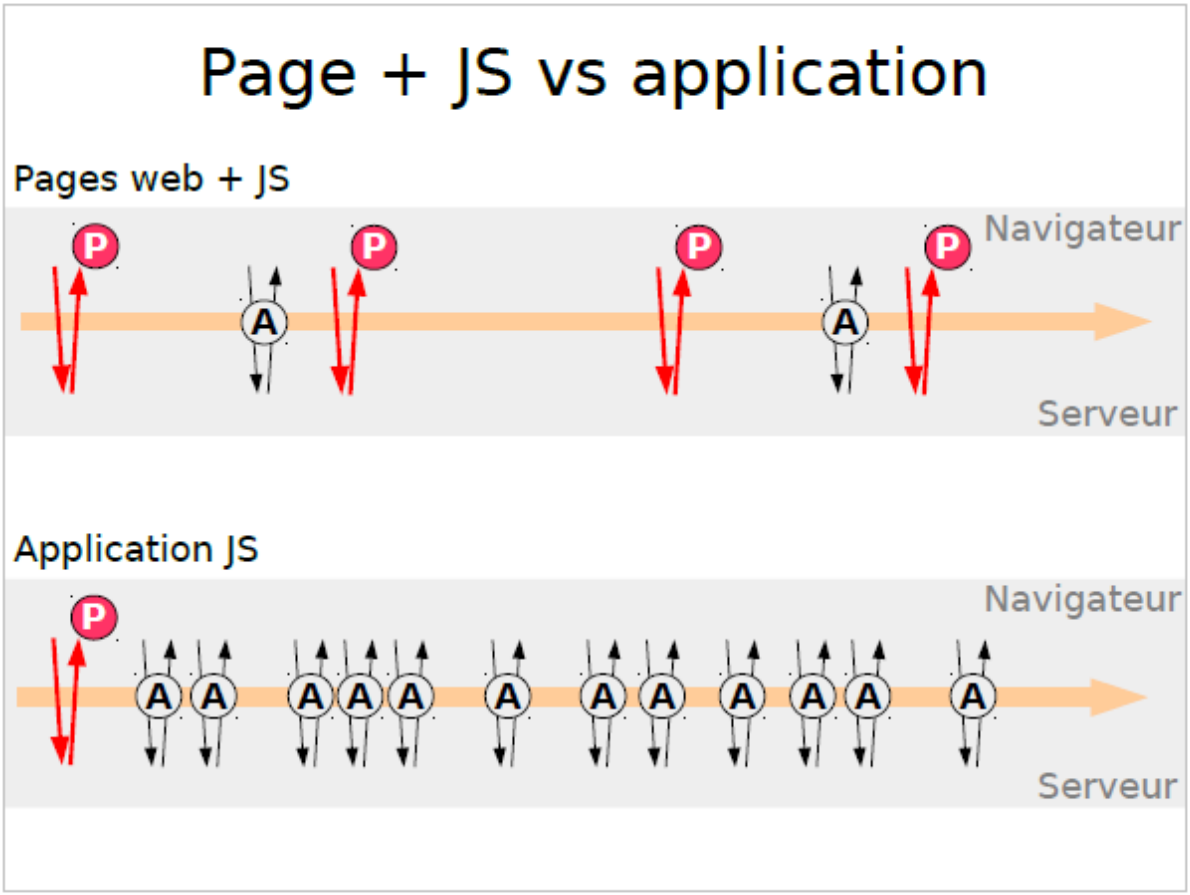
Voici un exemple simplifié.

Le client demande des données sur l'utilisateur numéro 1234.

Le PHP cherche ces données dans la BDD, les récupère dans un tableau associatif et transforme ce tableau en JSON.

Le JSON est renvoyé au client. JQuery décode le JSON et met l'objet décodé en argument (reponse) de la fonction anonyme.

Remarquez que jQuery est capable de distinguer automatiquement une réponse HTML d'une réponse JSON grâce au **Content-Type**.
Si Content-Type est text/html, **reponse** sera une chaîne de caractères contenant le HTML.
Si c'est application/json, **reponse** est un objet JS.



Dans l'approche classique (page web et JS), la navigation se fait en cliquant sur des liens ou en appuyant sur des boutons de formulaires.

A chaque fois une nouvelle page est visitée. De temps en temps une interaction AJAX peut venir enrichir cette navigation.

À l'autre extrême, dans une **application JS**, **une seule page** est chargée et l'utilisateur ne la quitte pas. L'utilisateur interagît avec cette page de manière complexe.
Ces interactions sont gérées en JS avec des appels AJAX fréquents. C'est ce que nous verrons lors de l'apprentissage du Framework **Angular**.

Un exemple d'application JS est **gmail**. L'utilisateur change de boite mail et ouvre chaque mail sans jamais quitter la page. Toutes ces **interactions** sont gérées en **JS/AJAX**.

Les **applications JS sont difficiles à écrire**. Le développeur doit gérer de nombreuses interactions complexes avec l'utilisateur et synchroniser les données avec le serveur.

Des Framework JS, comme **Angular** existent pour faciliter cette tâche. Ces techniques sortent du cadre de ce cours.

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services
Et l'appui du professeur M. BOSC de l'université Paris 13.
Document sous License Cnu Fdl.

Equipe de conception (IF, formateur, mediatiseur)
Formateur : Alexandre RESTOUEIX

Date de mise à jour : 29/01/19

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »