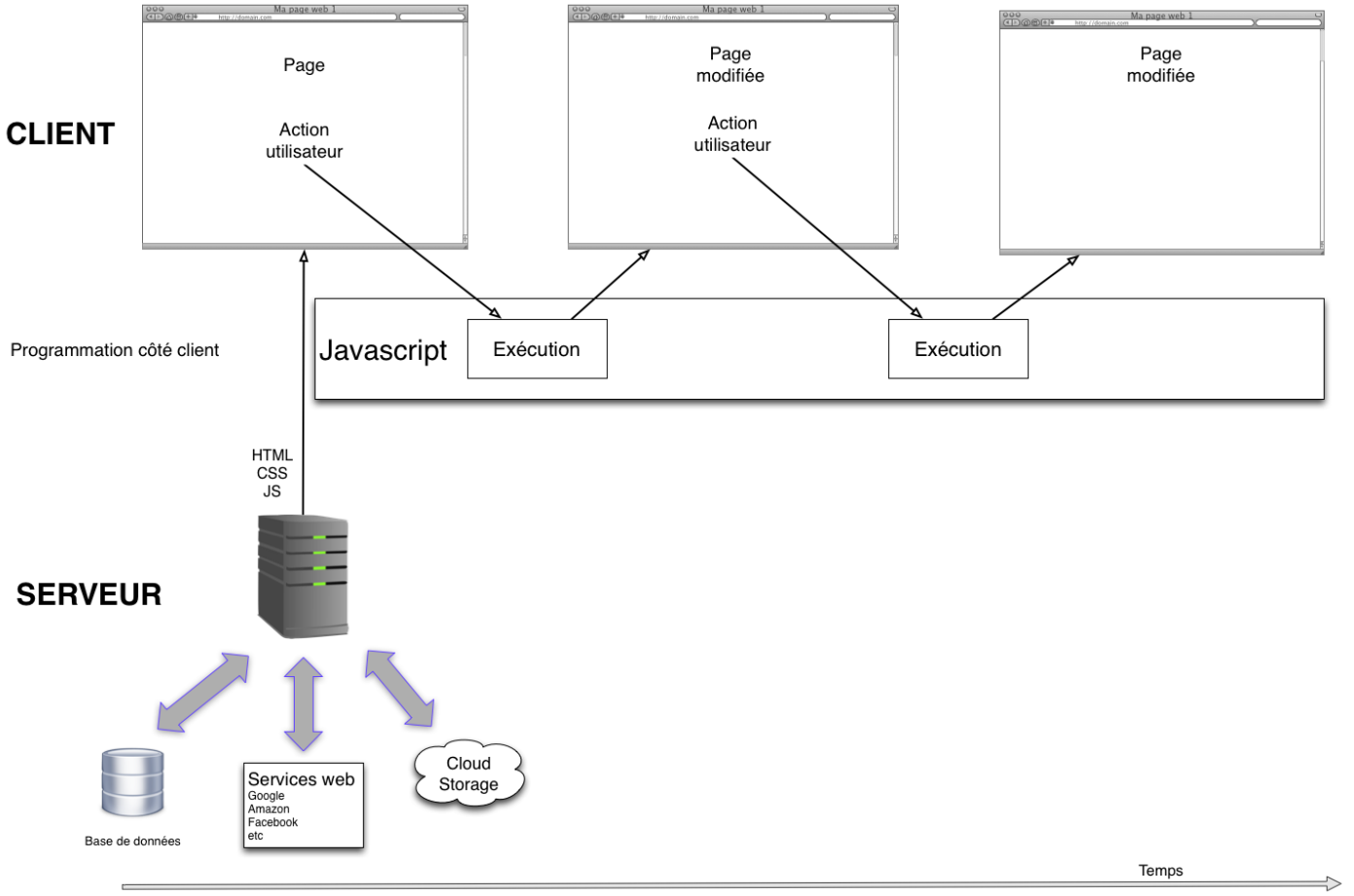
JS Asynchrone et APIs

Jean-Marc Lecarpentier

GREYC — Université de Caen

Des parties reprises du cours Ajax de Alexandre Niveau

Application client



Serveur fournit HTML/CSS et programmes JS
Interactions gérées en JS sur le client

Limites

- JavaScript permet de créer des pages interactives : les actions de l'internaute modifient la page
- Cependant, ça ne suffit pas toujours :
  - rafraîchissement automatique (webmail, chat…)
  - autocomplétion, suggestions
  - « exploration » (cartes, *infinite scrolling*…)
  - contenu modifiable

    - notes, votes, « likes », etc.
- Pages interactives qui communiquent avec le serveur

Idée

- JavaScript doit donc communiquer directement avec le serveur :
    - gérer la connexion HTTP et l'envoi de données GET ou POST
    - gérer la réception de la réponse du serveur
- Tout cela est géré grâce à une API : `XMLHttpRequest`

XMLHttpRequest

- Interface d'origine développée par Microsoft pour leur webmail Outlook Web Access
- Incorporée ensuite dans IE 5.0 (sorti en mars 1999)
- Copiée par Mozilla, implémentée comme un objet JS `XMLHttpRequest` dans Mozilla 0.6 (décembre 2000)
- Cette version devient un standard *de facto*
- Normalisé par le W3C à partir de 2006 [http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/], puis XMLHttpRequest level 2 à partir de 2008
- Depuis 2011, plus qu'un seul standard [http://www.w3.org/TR/XMLHttpRequest/], implémenté dans la plupart des navigateurs (IE > 9, voir tableau de compatibilité [http://caniuse.com/#search=xhr2])
- Initialement prévu pour travailler avec XML (d'où son nom), mais désormais majoritairement utilisée avec du JSON
- XHR

Principe de XHR

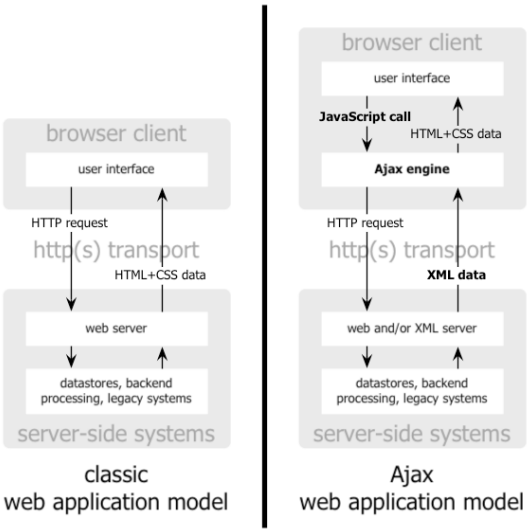
- On crée une requête sous la forme d'un objet XHR
- Requête HTTP a différents états (en cours, chargement, terminée, etc)
- Objet XHR déclenche des évènements lors des changements d'état de la requête
- On ajoute donc des capteurs d'évènements associés à des *callbacks* pour l'objet XHR
- On envoie la requête
- Les fonctions *callbacks* seront appelées en fonction des changements d'état de l'objet XHR
- N.B. : la réponse n'est pas forcément du XML... et le protocole pas forcément du HTTP.
- Attention : il faut ajouter les listener *avant* de lancer la requête !

Évènements XHR

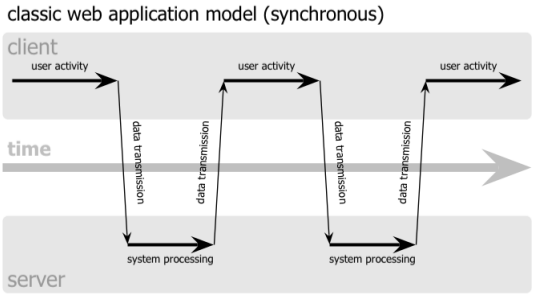
- `load` : la requête est terminée (avec succès) et toute la réponse a été reçue (on peut donc traiter la réponse)
- `error` : la requête a échoué (par ex. erreur 404)
- `timeout` : le temps maxi d'attente de la réponse est dépassé
- `abort` : lorsque la reqête est annulée
- `progress` : évènement déclenché à intervalles réguliers pour avoir des infos sur l'avancement du téléchargement de la réponse
- `loadend` : la requête est terminée (quelle que soit l'issue, succès ou échec)
- `loadstart` : le chargement des données commence
- `readystatechange` : à chaque changement d'état de la requête (historique, désormais inutile en général)

Ajax

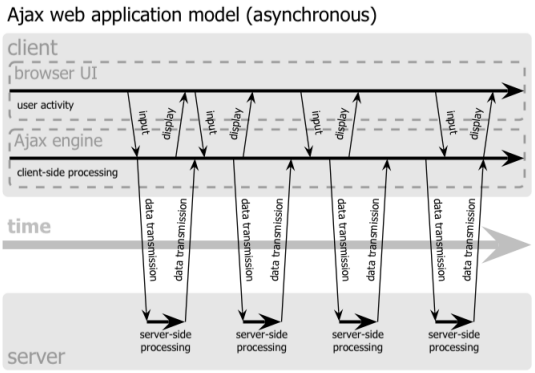
- L'utilisation de XMLHttpRequest (XHR) permet un nouveau modèle des interactions client-serveur
- Terme *Ajax* apparu début 2005 : Asynchronous Javascript and XML
- Principe : au lieu d'être synchrone, la communication client-serveur est asynchrone
- Initialement pensé pour fonctionner avec XML, Ajax est devenu un terme générique qui désigne l'ensemble des technologies nécessaires aux applications web asynchrones
- Sauvegarde de l'article original : Ajax: a New Approach to Web Applications, de Jesse James Garrett (18/02/2005) [https://web.archive.org/web/20050222032831/http://www.adaptivepath.com/publications/essays/archives/000385.php] (sur archive.org [https://archive.org/])



[images/ajax-fig1.png]



[images/ajax-sync.png]



[images/ajax-async.png]
Modèle classique du web comparé au modèle Ajax.
Source : Jesse James Garett

Utilisation de XHR

1. Créer un objet XHR
   `let requete = new XMLHttpRequest();`
2. Indiquer la méthode HTTP (GET en général) et l'URL à utiliser
   `requete.open("GET", "adresse_ressource");`
3. Mettre le(s) capteur(s) d'évènements
   `requete.addEventListener("load", traitementReponse);`
4. Spécifier le type de réponse attendu
   `requete.responseType = "type_de_réponse"`
5. Envoyer la requête
   `requete.send();`
6. Traitement de la réponse

```
function traitementReponse(event) {
    let req = event.currentTarget;
    console.log(req.response);
}
```

Type de réponse

- Préviser le type de la réponse attendu **avant** d'envoyer la requête
  `requete.responseType = "type_de_réponse"`
- `responseType` détermine le format des données qui sera contenu dans la propriété `response`
- Valeur par défaut vide (format texte brut dans ce cas)
- Types possibles :
  - `text` format texte brut (valeur "" aussi)
  - `json` format JSON qui sera parsé pour donner un objet Javascript
  - `document` format HTML ou XML qui sera parsé pour donner un arbre DOM (éventuellement partiel)

- `blob` réponse donnera un objet Blob
- `arraybuffer` réponse donnera un objet de type ArrayBuffer contenant les données binaires

Contenu de la réponse

- Dans tous les cas, la réponse sera contenue dans la propriété `response` au format défini par `responseType` (sauf erreur)
- **Attention** le terme *asynchrone* implique qu'on ne peut pas savoir quand la réponse arrivera (connexion lente, serveur saturé, etc), voir démos
- **Attention** si le format est `document` alors la réponse est un objet de type *Document*, on ne peut pas le mettre dans le DOM de la page.
  On obtient l'élément racine avec `requete.response.documentElement` (valable pour du XML ou du HTML)
  ou pour un document HTML son élément body avec `requete.response.body`
- Note : pour des raisons de compatibilité antérieure il existe aussi des propriétés responseText et responseXML qui sont **obsolètes, ne pas les utiliser**

Autres propriétés de XMLHttpRequest

- status : le code HTTP renvoyé par la requête
- statusText : la chaîne correspondant au code HTTP
- overrideMimeType(String mimetype) : force le type MIME de la réponse
- setRequestHeader(String header, String value) : en-tête HTTP à envoyer

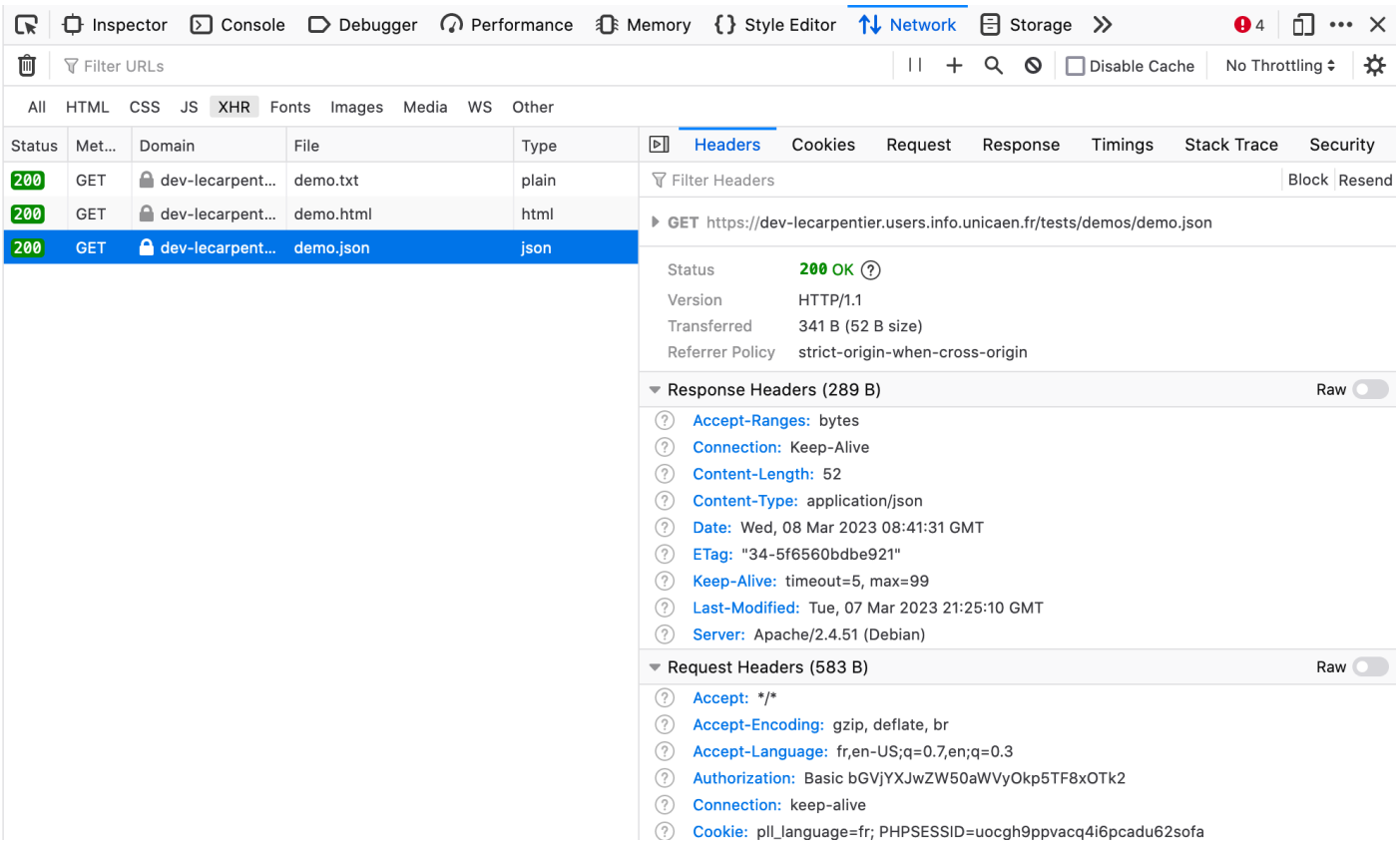
Requêtes POST

- Utilisation de l'interface FormData [https://developer.mozilla.org/en-US/docs/Web/API/FormData/Using_FormData_Objects]
- ```
  // créer un objet formData à partir du formulaire
  const data = new FormData(monFormulaire);
  const request = new XMLHttpRequest();
  request.open("POST", "/formHandler");
  // il suffit d'envoyer l'objet formData créé à partir du formulaire
  request.send(data);
  ```
- XHR combiné à l'API File [https://developer.mozilla.org/en-US/docs/Web/API/File] permet d'envoyer des fichiers sur un serveur
- Rien n'empêche également d'utiliser les autres méthodes de HTTP, comme PUT et DELETE

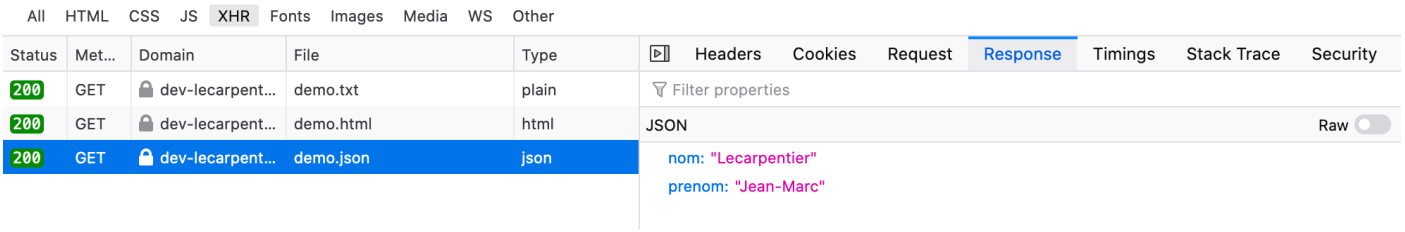
Exemples

Démos XHR [demos/01-xhr-simples.html]
  ```

Inspecter les requêtes XHR



Requêtes XHR en console



Visualisation de la réponse XHR

Références et guides

- Spécification de XHR [https://xhr.spec.whatwg.org/#interface-xmlhttprequest]
- Spécification de CORS [https://fetch.spec.whatwg.org/#http-cors-protocol]

Tutoriels

- Guide d'utilisation de XHR sur MDN [https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest]

Lectures complémentaires

- Ajax: a New Approach to Web Applications, de Jesse James Garrett (18/02/2005) [https://web.archive.org/web/20050222032831/http://www.adaptivepath.com/publications/essays/archives/000385.php]

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