Quelques APIs du Web

Michel Gagnon et Nikolay Radoev
École Polytechnique de Montréal

Les API du Web

- Le Web et les navigateurs sont composés des implémentations de multiples APIs définis par un standard (souvent HTML Living Standard ou ECMAScript)
- Nous avons vu le DOM API, mais voici quelques autres APIs utiles (ceux en gras seront vus dans le cours) :
  - Canvas API (rendu graphique 2D)
  - WebGL API (rendu graphique 2D ou 3D)
  - WebRTC (communication en temps réel)
  - **Web Storage API (stockage local d'information)**
  - **Fetch API (récupération de ressource à travers le réseau)**

Stockage local

- Deux possibilités:
  - L'information disparaît après la fin de la session, c'est-à-dire lorsque l'onglet contenant la page est fermé, ou lorsque le navigateur est fermé (objet **sessionStorage**)
    - Chaque onglet avec le même URL a son propre sessionStorage
  - L'information est persistante (objet **localStorage**)
- Dans les deux cas, on stocke des paires *attribut/valeur*
- Les objets sont accessibles à partir de l'objet global **window**
- L'espace de stockage est associé au domaine du site web (deux pages du même domaine peuvent accéder à la même information du stockage local

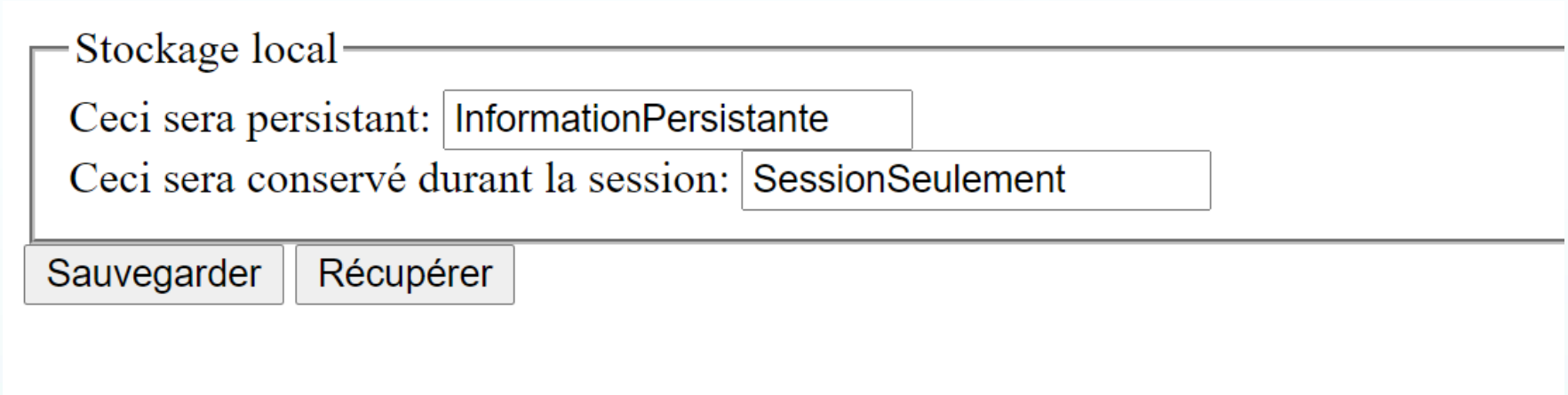
Stockage local – exemple simple

```
<!DOCTYPE html>
<html lang="en">
<head>…</head>
<body>
  <fieldset>
    <legend>Stockage local</legend>
    <label for="localData"> Ceci sera persistant:</label>
    <input id="localData"><br>
    <label for="sessionData"> Ceci sera conservé durant la session:</label>
    <input id="sessionData">
  </fieldset>
  <div>
    <button onclick="saveData()">Sauvegarder</button>
    <button onclick="loadData()">Récupérer</button>
  </div>
</body>
</html>
```

Stockage local – exemple simple

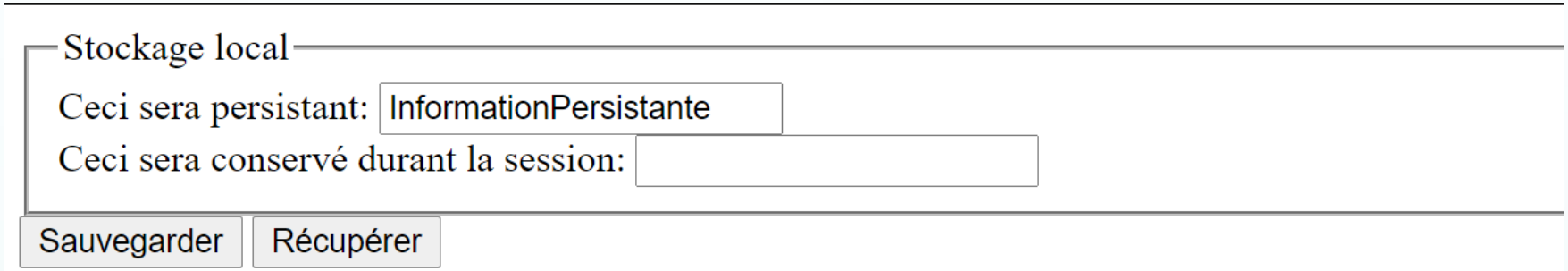
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <script>
    function saveData() {
       localStorage.setItem("localData", $("#localData").val());
       sessionStorage.setItem("sessionData", $("#sessionData").val());
    }
    function loadData() {
       if (localData != null) {
          $("#localData").val(localStorage.getItem("localData"));
       }
       if (sessionData != null) {
          $("#sessionData").val(sessionStorage.getItem("sessionData"));
       }
   }
 </script>
</head>
<body>…</body>
</html>
```

Stockage local – exemple simple



Adapté de *M. McDonald, HTML5, The Missing Manual, O'Reilly, 2014*

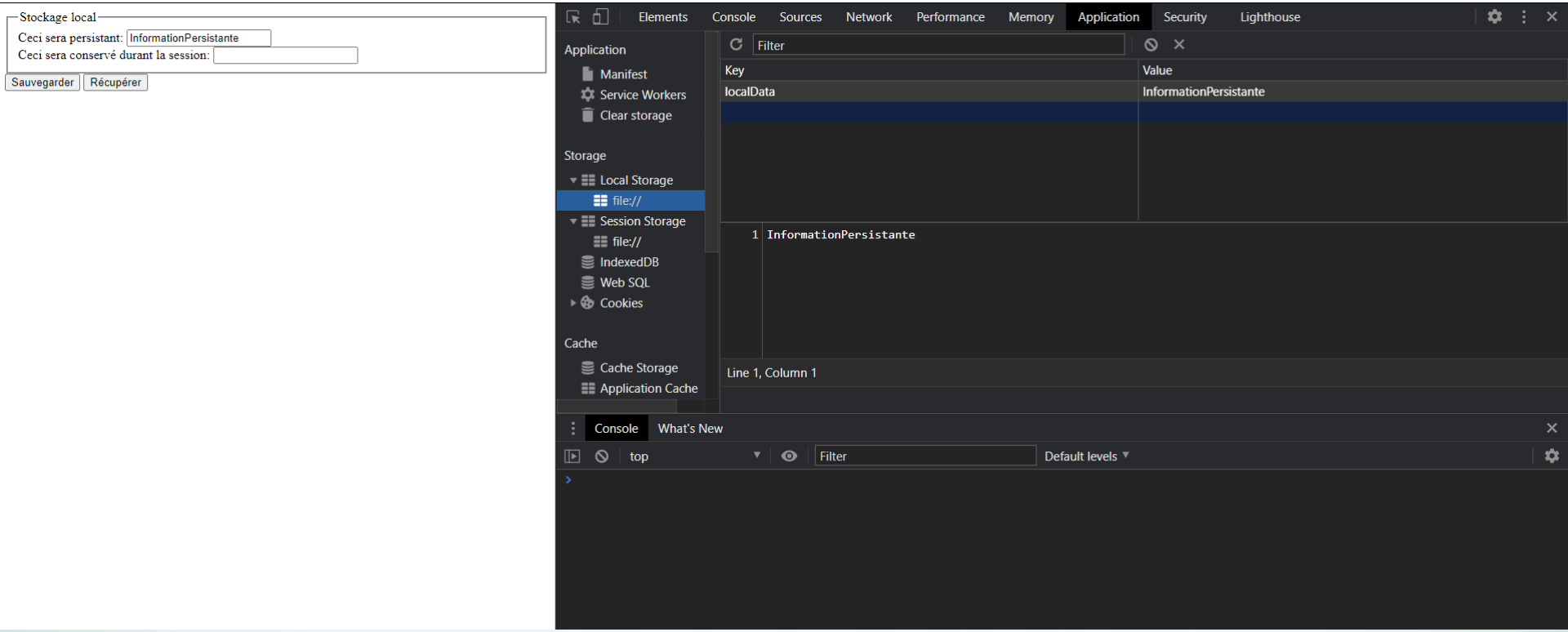
Stockage local – exemple simple



1. Cliquer sur "Sauvegarder"
2. Fermer la fenêtre
3. Ouvrir la page à nouveau
4. Cliquer sur "Récupérer"

Adapté de *M. McDonald, HTML5, The Missing Manual, O'Reilly, 2014*

Stockage local – exemple simple



Local Storage dans Chrome (onlget "Application" dans les outils

Adapté de *M. McDonald, HTML5, The Missing Manual, O'Reilly, 2014*

AJAX

- Principe: communiquer avec le serveur pour obtenir des données qui seront utilisées (en général) pour modifier le document sans recharger la page
- Deux manières courantes de l'implémenter:
  - Utilisation de l'objet **XMLHttpRequest**
  - Utilisation d'un élément **<script>**
- Alternative récente : Fetch API (encore en évolution)
- Technique asynchrone : Asynchronous JavaScript And XML
  - Contrairement au nom, on peut chercher plus de fichiers que du XML (le nom est plutôt historique)
- En général, le serveur envoie le contenu demandé dans un des formats suivants: texte, HTML, XML, JSON

XMLHttpRequest

1. On crée une instance de XMLHttpRequest
2. On attache un gestionnaire à l'événement **readystatechange**
    1. Si on veut juste la fin de la requête, on peut écouter l'événement **onload**
3. On ouvre la connexion (méthode **open()**)
4. On envoie la requête (méthode **send()**)
5. Dans le gestionnaire, on teste la valeur de l'attribut **readystate**, et on traite les informations reçues lorsque la valeur de cet attribut est **4**

XMLHttpRequest – Exemple GET

```
<script>
  const xhr = new window.XMLHttpRequest()
  xhr.open('GET', 'http://localhost:3000/comments')
  xhr.addEventListener('readystatechange', function () {
  // XMLHttpRequest.DONE a la valeur 4
  if (xhr.readyState === window.XMLHttpRequest.DONE
      && xhr.status === 200) {
      const comments = JSON.parse(xhr.responseText)
      // Faire quelque chose avec la variable ...
    }
  })
</script>
```

XMLHttpRequest – Exemple POST

```
<script>
  const xhr = new window.XMLHttpRequest()
  xhr.open('POST', 'http://localhost:3000/comments')
  xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')
  xhr.send(`auteur=${author}&commentaire=${comment}`)
  xhr.addEventListener('readystatechange', function () {
  if (xhr.readyState === window.XMLHttpRequest.DONE
      && xhr.status === 200) {
      const comment = JSON.parse(xhr.responseText)
      // Faire quelque chose avec la réponse du serveur
    }
  })
</script>
```

AJAX avec jQuery

- Méthode **load()**:
  - On lui passe une URL
  - Fait une reqûete HTTP GET à cette addresse
  - Le contenu envoyé par le serveur remplacera celui de l'élément sur lequel la méthode est exécutée
  - Exemple:

    $('#stats').load('rapport.html')

- Arguments supplémentaires possibles:
  - Un ensemble de données à ajouter à la requête
  - Une fonction qui sera exécutée lorsque le traitement de la requête aura été complété

AJAX avec jQuery

- Fonction utilitaire **$.getScript()**:
  - Télécharge et exécute un code Javascript
  - Pas régi par la contrainte de la même origine
- Fonction utilitaire **$.getJSON()**:
  - Fait une requête avec l'URL fournie
  - Parse la réponse JSON et transmet l'objet résultant à la fonction passée en paramètre
- Fonctions **$.get()** et **$.post()**:
  - Font une requête avec la méthode correspondante
- Fonction générique **$.ajax()**:
  - On lui passe un ensemble de données qui définissent le genre d'interaction désiré

AJAX avec jQuery - Exemple

```
<script>
  $.get('http://localhost:3000/comments', function(comments) {
    // FAIRE QUELQUE CHOSE AVEC LA VARIABLE
  })

  $.post('http://localhost:3000/comments',
          { author, comment},
          function(comment) {
    // FAIRE QUELQUE CHOSE AVEC LA VARIABLE
  })
</script>
```

AJAX avec jQuery - Exemple

```
<script>
  $.ajax({
    url: ‘/http://localhost:3000/comments/',
    type: 'DELETE’,                // On fait un HTTP DELETE
    success: function(result) {
        // Permet de traiter le résultat
    }
});</script>
```

Fetch API

- Alternative plus moderne (depuis 2015) pour chercher des ressources sur Web:
  - Peut chercher une ressource locale ou à travers le reseau (alternative intéressante à XMLHttpRequest)
- Basé sur les Promesses, contrairement aux événements de XHR
- Remplacement potentiel de XHR
- API encore en évolution
  - La possibilité d'annuler une requête *fetch* seulement rajouté en mars 2020 et encore expérimental

Fetch API

- Pour faire une requête, on utilise la fonction globale *fetch(resource[, init])*
  - *resource* est la ressource à chercher (souvent un URL)
  - *init* est un objet de configuration avec les spécifications du type de requête à envoyer. S'il est omis, *fetch*() fait une requête GET sur la *resource*
- La fonctionne retourne une promesse ou lance une exception
  - Note: un code HTTP 400 ou 500 n'est pas considéré comme une erreur par *fetch()*

Fetch API – Exemple simple

```
<script>
    // on fait un GET par défaut
    fetch('http://example.com/movies.json')
    .then(response => response.json())
    .then(data => console.log(data));});
</script>
```

Fetch API – Exemple plus poussé

```
// Example POST method implementation:
async function postData(url = '', data = {}) {
  // Default options are marked with *
  const response = await fetch(url, {
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *cors, same-origin
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    credentials: 'same-origin', // include, *same-origin, omit
    headers: {
      'Content-Type': 'application/json'
    },
    redirect: 'follow', // manual, *follow, error
    referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-
downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-
cross-origin, unsafe-url
    body: JSON.stringify(data) // body data type must match "Content-Type" header
  });
  return response.json(); // parses JSON response into native JavaScript objects
}

postData('https://example.com/answer', { answer: 42 })
  .then(data => {
    console.log(data); // JSON data parsed by `data.json()` call
  });
```