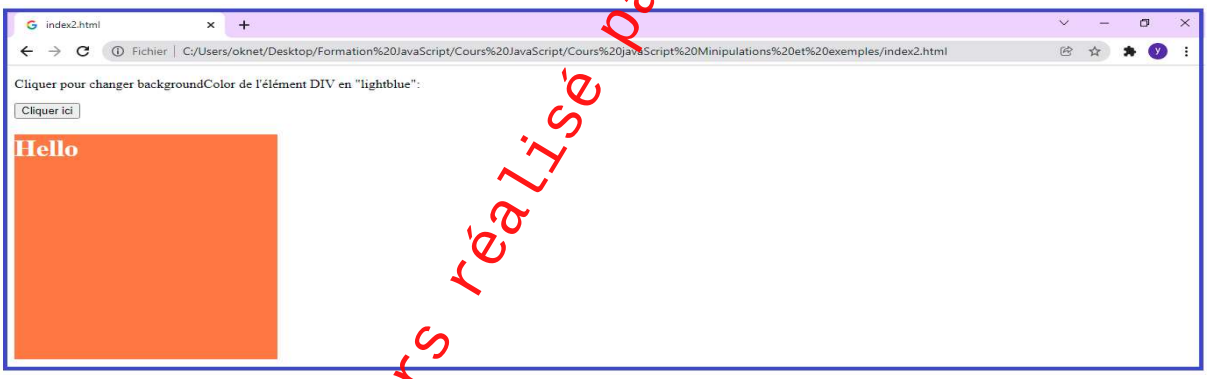
- Exemple pour changer plusieurs propriétés :

```
<!DOCTYPE html>
<html>
<head>
    <style>
        #myDIV {
            width: 300px;
            height: 300px;
            background-color: coral;
            color: white;
        }
    </style>
</head>
<body>
    <p>Cliquer pour changer l'élément DIV :</p>
    <button onclick="myFunction()">Cliquer ici</button>
    <div id="myDIV">
        <h1>Hello</h1>
    </div>
    <script>
        function myFunction() {
            var div = document.getElementById("myDIV");
            div.style.backgroundColor="lightblue";
            div.style.width="100px";
            div.style.height="100px";
            div.style.border="3px solid black";
            div.style.color= "black";
        }
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat



- Après avoir cliqué sur le bouton on obtient

Evènements :

Un évènement est provoqué par une action de l'utilisateur ou du navigateur lui-même. Les évènements ont des noms tels que **click**, **load** et **mouseover**.

Une fonction qui est appelée en réponse à un évènement est nommée **un écouteur** (**event handler** ou **event listener**). Souvent, son nom commence par **on** comme par exemple **onclik** ou **onload**.

Associer des écouteurs aux évènements possibles peut se faire de trois manières différentes :

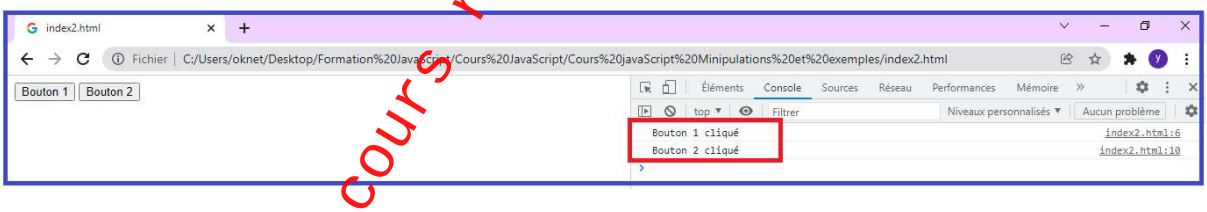
- HTML
- DOM Level 0
- DOM Level 2

- **HTML Event Handlers :**

On utilise des attributs HTML pour déclarer les écouteurs. La valeur de ces attributs est le code JavaScript à exécuter lorsque l'évènement est produit.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <button onclick="console.log('Bouton 1 cliqué')">Bouton 1</button>
    <button onclick="showMessage()">Bouton 2</button>
    <script>
        function showMessage() {
            console.log('Bouton 2 cliqué');
        }
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat



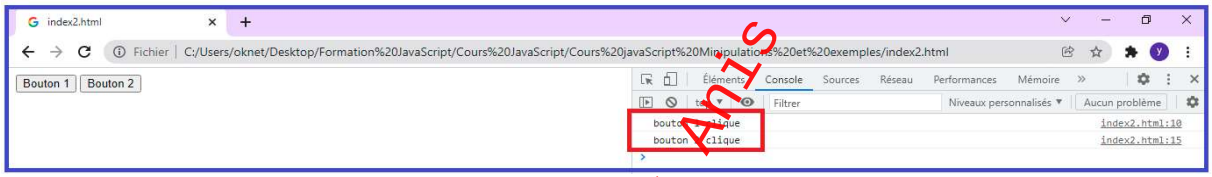
- **Dom Level 0 Event Handlers :**

On utilise les propriétés des éléments pour leur associer des écouteurs.

```
<!DOCTYPE html>
<html>
 <head>
 </head>
 <body>
    <button id="but1">Bouton 1</button>
    <button id="but2">Bouton 2</button>
    <script>
        function but1Listener() {
            console.log("bouton 1 cliqué");
        }
        let but1 = document.getElementById("but1");
        but1.onclick = but1Listener;

        document.getElementById("but2").onclick = function () {
            console.log("bouton 2 cliqu\é");
        }
    </script>
 </body>
</html>
```

- Tester au navigateur pour voir le résultat



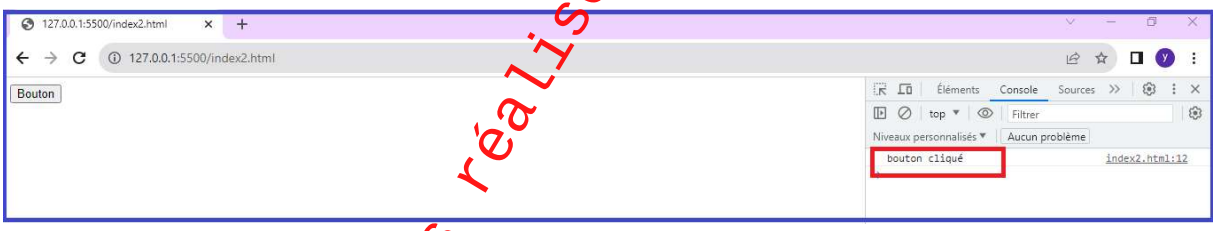
- **Remarque :**

Quand un évènement se produit, toutes les informations le concernant sont enregistrées dans un objet. Il est possible de récupérer cet objet comme paramètre d'une fonction écouteur.

```
<!DOCTYPE html>
<html>
 <head>
     <meta charset="utf-8">
 </head>
 <body>
    <button id="btn">Bouton</button>
    <script>
       function listener(evt) {
             switch(evt.type) {
              case "click" :
                    console.log("bouton cliqué");
                    break;
              case "mouseover" :
                    this.style.backgroundColor="red";
                    break;
              case "mouseout" :
                    this.style.backgroundColor="";
              }
        }

        let btn = document.getElementById("btn");
        btn.onclick=listener;
        btn.onmouseover=listener;
        btn.onmouseout=listener;
    </script>
 </body>
</html>
```

- Tester au navigateur pour voir le résultat



- **Dom Level 2 Event Handlers :**

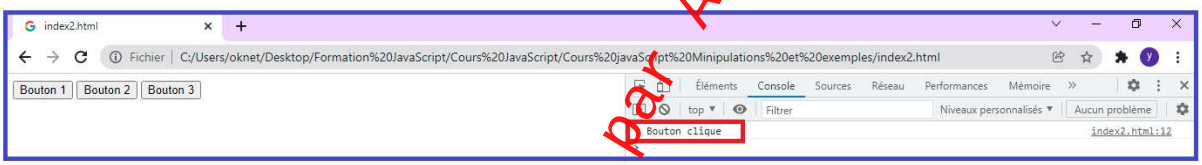
On utilise la méthode **addEventListener** qui permet d'associer des écouteurs à un élément cible. La cible peut être **un nœud dans un document**, **le document lui-même**, **un élément window** ou **un objet XMLHttpRequest**.

La méthode prend généralement deux paramètres :

- ❖ le type de l'évènement : tel que '**click**', '**mousedown**', '**mouseup**', ...
- ❖ l'écouteur : un objet implantant **EventListener**, ou une **fonction**,

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
</head>
<body>
   <button id="but1">Bouton 1</button>
   <script>
      let src = document.querySelector('button'); // cible tous les éléments <button>
      src.addEventListener("click", function() {
              console.log('Bouton cliqué');
      });
   </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat



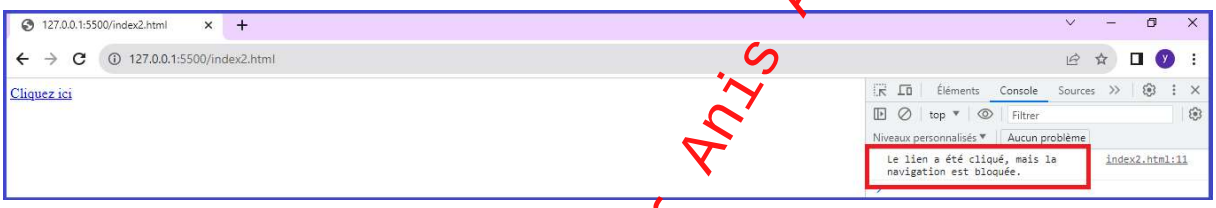
- **Contrôle et Catégories :**

Parmi les instructions permettant de contrôler les évènements :

- **preventDefault :** Empêche le comportement par défaut d'un élément. Par exemple, elle peut empêcher un lien de naviguer vers une nouvelle page lorsque celui-ci est cliqué.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <a href="https://www.example.com" id="lien">Cliquez ici</a>
    <script>
        const lien = document.querySelector('#lien');
        lien.addEventListener('click', (event) => {
            event.preventDefault();
            console.log('Le lien a été cliqué, mais la navigation est bloquée.');
        });
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat



❖ **stopPropagation :** Empeche la propagation de l'événement vers les éléments parents. Cela évite que des événements soient déclenchés sur plusieurs niveaux d'éléments imbriqués.

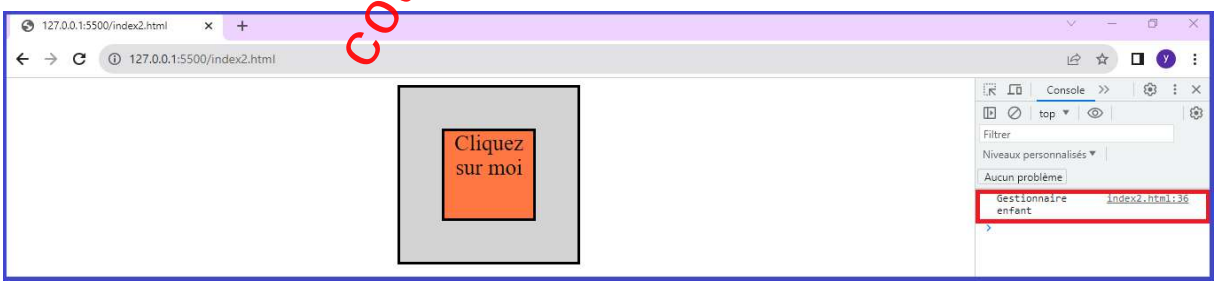
```
<!DOCTYPE html>
<html>
<head>
    <style>
        .parent{
            background-color: yellow;
        }
    </style>
</head>
<body>
    <div class="parent">
        <button class="enfant">Cliquez sur le bouton enfant</button>
    </div>
    <script>
        const divParent = document.querySelector('.parent');
        const boutonEnfant = document.querySelector('.enfant');
        divParent.addEventListener('click', () => {
            console.log('Div parent cliqué.');
        });
        boutonEnfant.addEventListener('click', (event) => {
            event.stopPropagation();
            console.log('Bouton enfant cliqué, la propagation est stoppée.');
        });
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat

❖ **evt.stopImmediatePropagation() :** Stopper l'événement en cours et empêcher d'autres actions liées à cet événement de se déclencher.

```
<!DOCTYPE html>
<html>
<head>
<style>
  .parent {
    width: 200px;
    height: 200px;
    background-color: lightgray;
    border: 3px solid black;
    display: flex;
    justify-content: center;
    align-items: center;
    margin:auto;
  }
  .enfant {
    width: 100px;
    height: 100px;
    background-color: coral;
    border: 3px solid black;
    text-align: center;
    font-size: 25px;
  }
</style>
</head>
<body>
  div class="parent">
      <div class="enfant">Cliquez sur moi</div>
  </div>
  <script>
      const parent = document.querySelector('.parent');
      const enfant = document.querySelector('.enfant');
      parent.addEventListener('click', () => {
          console.log('Gestionnaire parent');
      });
      enfant.addEventListener('click', (event) => {
         console.log('Gestionnaire enfant');
         event.stopImmediatePropagation();
         // Arrête la propagation et empêche les autres gestionnaires d'être exécutés
      });
  </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat



- **removeEventListener** : Permet de supprimer un écouteur d'événements précédemment attaché à un élément. Cela permet de désactiver la réaction à un événement spécifique.

```html
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <button id="bouton">Cliquez ici</button>
    <script>
        const bouton = document.querySelector('#bouton');
        function gestionnaireClic() {
            console.log('Le bouton a été cliqué.');
        }
        bouton.addEventListener('click', gestionnaireClic);
        // Ajouter un timer de 4 secondes avant de supprimer l'écouteur d'événements
        setTimeout(() => {
            bouton.removeEventListener('click', gestionnaireClic);
            console.log("L'écouteur d'événements a été supprimé après 4 secondes.");
        }, 4000); // 4000 millisecondes équivalent à 4 secondes
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat

Catégories des évènements prédéfinis :

- **Evènements souris :**
  - **Click**
  - **dblclick**
  - **MouseDown :** lorsqu'on enfonce un bouton de la souris sur un élément.
  - **MouseUp :** lorsqu'on relâche un bouton de la souris après avoir cliqué sur un élément.
  - **MouseOver :** lorsque le curseur de la souris entre dans la zone d'un élément.
  - **MouseOut :** lorsque le curseur de la souris quitte la zone d'un élément.
  - **MouseMove :** lorsque le curseur de la souris se déplace sur un élément.

Les propriétés accessibles à partir de l'objet **event** de ces évènements sont :

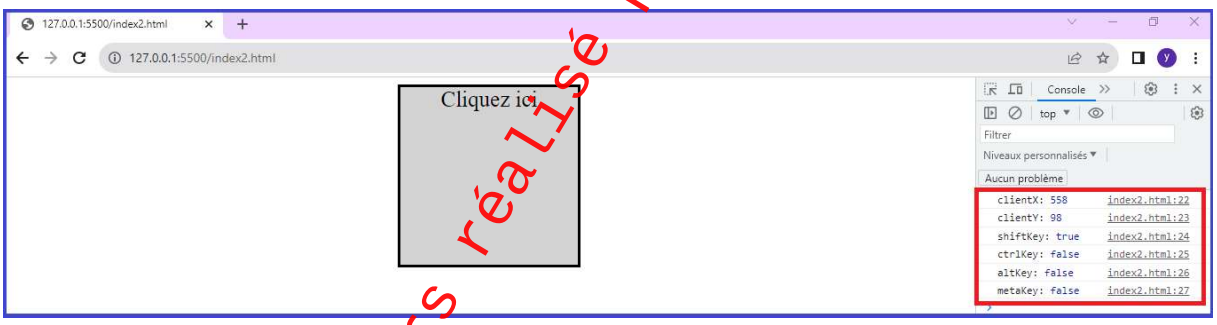
- **clientX :** La position horizontale du curseur dans la fenêtre du navigateur lorsque le clic a été effectué.
- **clientY** : La position verticale du curseur dans la fenêtre du navigateur lorsque le clic a été effectué.
- **shiftKey :** Indique si la touche "Shift" a été enfoncée lors du clic.
- **ctrlKey :** Indique si la touche "Ctrl" a été enfoncée lors du clic.
- **altKey :** Indique si la touche "Alt" a été enfoncée lors du clic.
- **metaKey :** Indique si la touche "Meta" (Command sur Mac ou Windows sur PC) a été enfoncée lors du clic.

Exemple :

```html
<!DOCTYPE html>
<html>
<head>
    <style>
        #zone{
            width: 200px;
            height: 200px;
            background-color: lightgray;
            border: 3px solid black;
            margin: auto;
            text-align: center;
            font-size: 25px;
        }

    </style>
</head>
<body>
    <div id="zone">Cliquez ici</div>
    <script>
        const zone = document.querySelector('#zone');
        zone.addEventListener('click', (event) => {
            console.log('clientX:', event.clientX);
            console.log('clientY:', event.clientY);
            console.log('shiftKey:', event.shiftKey);
            console.log('ctrlKey:', event.ctrlKey);
            console.log('altKey:', event.altKey);
            console.log('metaKey:', event.metaKey);
        });
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat



- **Evènements du clavier :**
  - **KeyDown :** Se produit lorsqu'on enfonce une touche du clavier.
  - **KeyUp :** Se produit lorsqu'on relâche une touche du clavier après l'avoir enfoncée.
  - **KeyPress :** Se produisait lorsqu'une touche du clavier est enfoncée et maintenue, mais il est recommandé d'utiliser keydown à sa place pour des raisons de compatibilité avec les navigateurs modernes.

Les propriétés utiles et accessibles à partir de l'objet **event** de ces évènements sont :

- **key :** Contient la valeur de la touche du clavier qui a déclenché l'événement.
- **keyCode :** Donne le code numérique de la touche du clavier qui a été enfoncée.
- **ctrlKey :** Indique si la touche "Ctrl" a été enfoncée en même temps que la touche du clavier.
- **altKey :** Indique si la touche "Alt" a été enfoncée en même temps que la touche du clavier.
- **shiftKey :** Indique si la touche "Shift" a été enfoncée en même temps que la touche du clavier.
- **metaKey :** Indique si la touche "Meta" (Command sur Mac ou Windows sur PC) a été enfoncée en même temps que la touche du clavier.
- **repeat :** Indique si la touche du clavier est maintenue enfoncée et répète l'événement.

Exemple :

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <input id="inputField" type="text" placeholder="Appuyez sur une touche" />
    <script>
        const inputField = document.querySelector('#inputField');
        inputField.addEventListener('keydown', (event) => {
            console.log('Key:', event.key);
            console.log('KeyCode:', event.keyCode);
            console.log('Ctrl:', event.ctrlKey);
            console.log('Alt:', event.altKey);
            console.log('Shift:', event.shiftKey);
            console.log('Meta:', event.metaKey);
            console.log('Repeat:', event.repeat);
        });
    </script>
</body>
</html>
```

- Tester au navigateur pour voir le résultat

- **Autres Evènements :**
  - **Load :** Se produit lorsque la page ou une ressource est complètement chargée.
  - **Unload :** Se produit lorsque la page est en train d'être déchargée ou quittée.
  - **Abort :** Se produit lorsqu'un chargement de ressource est annulé.
  - **Error :** Se produit lorsqu'une erreur survient lors du chargement d'une ressource.
  - **Select :** Se produit lorsque du texte est sélectionné dans un champ de texte ou une zone de texte.
  - **Change :** Se produit lorsqu'une valeur dans un élément de formulaire (comme une case à cocher) est modifiée.
  - **Submit :** Se produit lorsque le formulaire est soumis.
  - **Reset :** Se produit lorsqu'un formulaire est réinitialisé.
  - **Resize :** Se produit lorsque la fenêtre du navigateur est redimensionnée.
  - **Scroll :** Se produit lorsque l'utilisateur fait défiler le contenu d'une fenêtre ou d'une zone de défilement.
  - **Focus :** Se produit lorsque l'élément obtient le focus (est sélectionné) par l'utilisateur.
  - **Blur :** Se produit lorsque l'élément perd le focus (n'est plus sélectionné) par l'utilisateur.

La plupart de ces évènements sont liés aux **formulaires** ou à l'objet **window**.

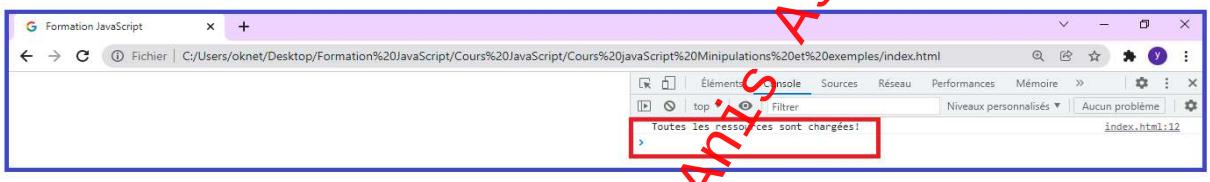
- **Evènements load et unload :**

Pour l'objet **window**, l'évènement **load** se produit lorsque la page complète est chargée, incluant les ressources externes telles que les images, le fichier JavaScript **script.js** et le fichier CSS **style.css**.

```
<!DOCType html>
<html lang="fr">
  <head>
     <meta charset="utf-8">
     <title>Formation JavaScript</title>
     <link href="styles.css" rel="stylesheet">
     <script  src="script.js" type="text/JavaScript" defer></script>
  </head>
  <body>
    <script>
      window.addEventListener("load", function(event){
                                    console.log("Toutes les ressources sont chargées!");
                                    });

    </script>
  </body>
</html>
```

- Tester le fichier index2.html au navigateur pour voir le résultat



- Il est possible d'associer cet évènement à des éléments **img**.

```
<!DOCType html>
<html lang="fr">
  <head>
     <meta charset="utf-8">
     <title>Formation JavaScript</title>
  </head>
  <body>
    <img id="img1" src="image.png" width="200" height="200">
     <script>
      document.getElementById("img1").addEventListener("load", function(){
                                    console.log("L'image est chargées!");
                                    });

     </script>
  </body>
</html>
```

- **Remarque**

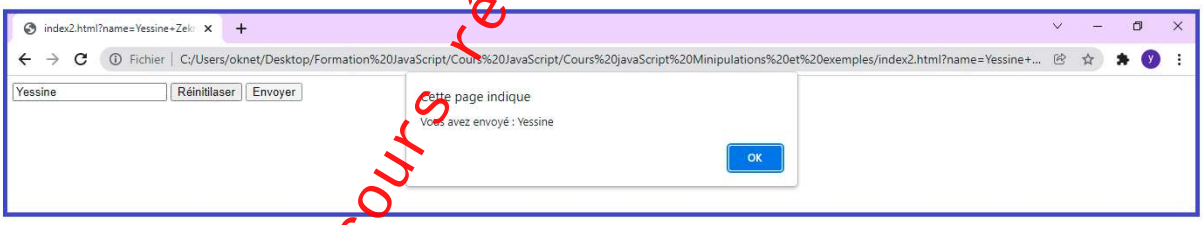
L'évènement **unload** se produit typiquement lorsqu'on change de page. Cela permet par exemple de libérer certaines ressources.

- **Evènements submit et reset :**

Ils sont associés à un formulaire.

```
<!DOCTYPE html>
<html>
 <head>
 </head>
 <body>
  <form action="#">
      <input name='name'>
      <button type='reset'>Réinitilaser</button>
      <button type='submit'>Envoyer</button>
  </form>
  <script>
    let form = document.querySelector('form');
    form.addEventListener("submit", function(evt)
                                    {
                                       if (form.name.value === 'toto')
                                       {
                                           evt.preventDefault();
                                           alert("Vous n'avez pas envoyé des données");
                                       }
                                       else
                                       {
                                           alert("Vous avez envoyé : " + form.name.value);
                                       }
                                    });
    form.addEventListener("reset", function(evt)
                                   {
                                      if (!confirm("Voulez-vous vider le formulaire?"))
                                           evt.preventDefault();
                                   });
  </script>
 </body>
</html>
```

- Tester pour voir le résultat



- **Evenements focus et blur :**

```
<!DOCTYPE html>
<html>
 <head>
 </head>
 <body>
   <input>
   <script>
    let e = document.querySelector('input');
    e.addEventListener("focus", function(evt){
                                            if(e.style.backgroundColor !== "red")
                                                e.style.backgroundColor="yellow";
                                          });
    e.addEventListener("blur", function(evt){
                                            if(e.value.match(/\d/g)) //des chiffres
                                               e.style.backgroundColor = "red";
                                            else
                                               e.style.backgroundColor = "green";
                                          });
   </script>
 </body>
</html>
```

- Tester pour voir le résultat

Gestion des Formulaires en JavaScript

La gestion des formulaires en JavaScript permet d'interagir avec les éléments de formulaire d'une page web. On peut accéder aux valeurs saisies par les utilisateurs, effectuer des validations et réagir aux événements liés aux formulaires. Voici comment procéder avec des exemples simples.

- **Accéder aux Valeurs de Formulaire :**

  Accédez aux valeurs saisies par l'utilisateur dans les champs de formulaire.

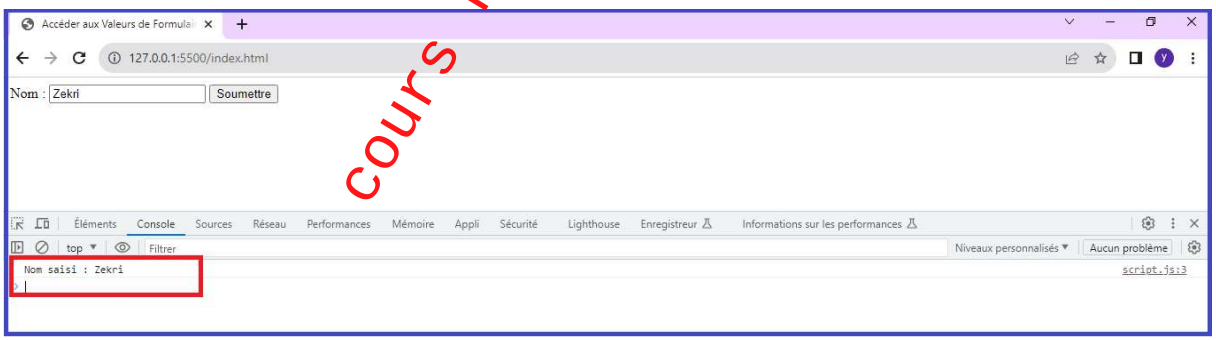
  Copier le code suivant dans le fichier index2.html :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Accéder aux Valeurs de Formulaire</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
  <form id="myForm">
    <label for="name">Nom :</label>
    <input type="text" id="name" name="name" required>
    <button type="button" onclick="getFormValues()">Soumettre</button>
  </form>
</body>
</html>
```

  Copier le code suivant dans le fichier script.js :

```javascript
function getFormValues() {
  const name = document.getElementById('name').value;
  console.log("Nom saisi :", name);
}
```

- Tester au navigateur pour voir le résultat :



- **Validation de Formulaire :**

  Effectuez des validations de base sur les champs de formulaire.

  Copier le code suivant dans le fichier index2.html :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Validation de Formulaire</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
  <form id="myForm" onsubmit="return validateForm()">
    <label for="age">Âge :</label>
    <input type="number" id="age" name="age" required>
    <button type="submit">Soumettre</button>
  </form>
</body>
</html>
```

  Copier le code suivant dans le fichier script.js :

```javascript
function validateForm() {
  const age = document.getElementById('age').value;
  if (age < 18) {
    alert("Vous devez avoir au moins 18 ans !");
    return false; // Empêche la soumission du formulaire
  }
  return true; // Permet la soumission du formulaire
}
```

- Tester au navigateur pour voir le résultat :

- **Réagir aux Événements de Formulaire :**

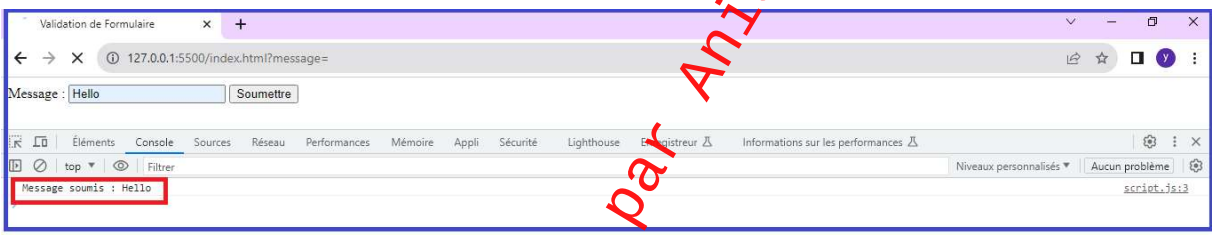
Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Réagir aux Événements de Formulaire</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
    <form id="myForm" onsubmit="submitForm()">
        <label for="message">Message :</label>
        <input type="text" id="message" name="message">
        <button type="submit">Soumettre</button>
      </form>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
function submitForm() {
  const message = document.getElementById('message').value;
  console.log("Message soumis :", message);
}
```

- Tester au navigateur pour voir le résultat :



- **Réinitialiser un Formulaire :**

Réinitialisez les valeurs d'un formulaire après soumission (fichier index.html).

```
<!DOCTYPE html>
<html>
<head>
  <title>Réinitialiser un Formulaire</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
    <form id="myForm" onsubmit="submitForm(); return false;">
        <label for="color">Couleur préférée :</label>
        <input type="text" id="color" name="color">
        <button type="submit">Soumettre</button>
    </form>
    <button onclick="resetForm()">Réinitialiser</button>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
function submitForm() {
  const color = document.getElementById('color').value;
  console.log("Couleur préférée :", color);
}
function resetForm() {
  document.getElementById('myForm').reset();
  console.log("Formulaire réinitialisé !");
}
```

- Tester au navigateur pour voir le résultat :

- **Action et Method d'un Formulaire :**

  L'attribut **action** définit l'URL de destination lors de la soumission du formulaire, et l'attribut **method** détermine la méthode HTTP utilisée (**GET** ou **POST**).

  Copier le code suivant dans le fichier index.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Action et Method d'un Formulaire</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
    <form id="myForm" action="traitement.php" method="post">
        <!-- ... Champs de formulaire ... -->
        <button type="submit">Soumettre</button>
    </form>
</body>
</html>
```

- Tester au navigateur pour voir le résultat :

- **Utilisation de document.forms :**

  L'objet **document.forms** permet d'accéder à une collection d'éléments de formulaire.

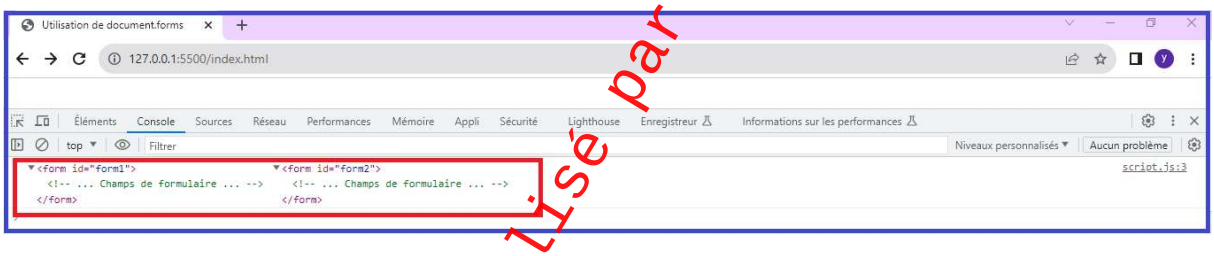
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
    <title>Utilisation de document.forms</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
    <form id="form1">
        <!-- ... Champs de formulaire ... -->
      </form>
      <form id="form2">
        <!-- ... Champs de formulaire ... -->
      </form>
</body>
</html>
```

  Copier le code suivant dans le fichier script.js :

```
const form1 = document.forms[0]; // Accède au premier formulaire
const form2 = document.forms["form2"]; // Accède au formulaire avec l'id "form2"
console.log(form1, form2);
```

- Tester au navigateur pour voir le résultat :



- **Accès aux Valeurs des Champs de Formulaire :**

  La propriété **.value** utilisé pour accéder aux valeurs saisies dans les champs.

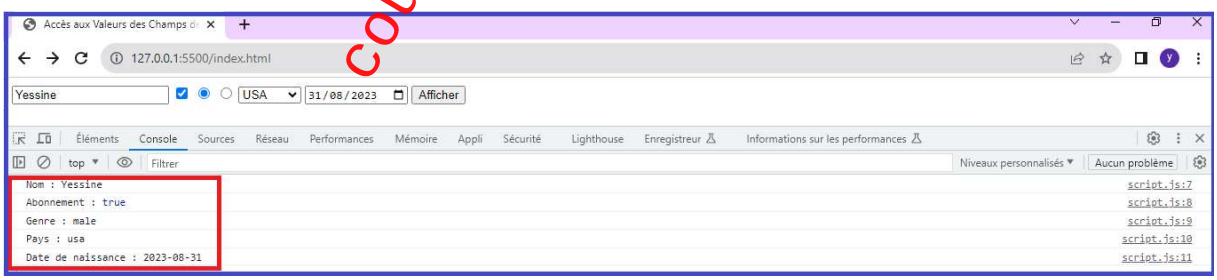
  Copier le code suivant dans le fichier index.html :

```
<!DOCTYPE html>
<html>
<head>
    <title>Accès aux Valeurs des Champs de Formulaire</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
    <form id="myForm">
        <input type="text" id="name" name="name" value="John">
        <input type="checkbox" id="subscribe" name="subscribe" checked>
        <input type="radio" id="male" name="gender" value="male">
        <input type="radio" id="female" name="gender" value="female" checked>
        <select id="country" name="country">
          <option value="usa">USA</option>
          <option value="canada" selected>Canada</option>
        </select>
        <input type="date" id="birthdate" name="birthdate">
        <button type="button" onclick="getFormValues()">Afficher</button>
      </form>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
function getFormValues() {
  const name = document.getElementById('name').value;
  const subscribe = document.getElementById('subscribe').checked;
  const gender = document.querySelector('input[name="gender"]:checked').value;
  const country = document.getElementById('country').value;
  const birthdate = document.getElementById('birthdate').value;
  console.log("Nom :", name);
  console.log("Abonnement :", subscribe);
  console.log("Genre :", gender);
  console.log("Pays :", country);
  console.log("Date de naissance :", birthdate);
}
```

- Tester au navigateur pour voir le résultat :



Conclusion : Les formulaires en HTML et JavaScript offrent un moyen essentiel d'interagir avec les utilisateurs en collectant, validant et traitant les données saisies.

Ajax

Ajax (Asynchronous JavaScript and XML) permet une communication asynchrone avec un serveur. Des éléments de la page peuvent être mis à jour avec les données du serveur (sans recharger toute la page). Il faut un serveur (à lancer avec node.js).

- Créer un serveur au fichier server.js

```
// Mode strict pour appliquer des règles de codage strictes.
'use strict';
// Importation du module 'http' pour créer un serveur HTTP.
const http = require('http');
// Création du serveur HTTP en utilisant la méthode createServer
const server = http.createServer(function(req, res) {
    // Configurer les en-têtes de la réponse pour indiquer le type de contenu JSON.
    res.setHeader('Content-Type', 'application/json');
    // Configurer l'en-tête permettant l'accès depuis n'importe où (CORS).
    res.setHeader('Access-Control-Allow-Origin', '*');
    // Construction de l'objet de réponse au format JSON
    const responseData = {
      Formation: 'JavaScript',
      platform: process.platform,          // Plate-forme du système d'exploitation.
      nodeVersion: process.version,        // Version de Node.js.
      uptime: Math.round(process.uptime()) // Durée d'exécution du serveur
    };
    // Envoi au client la réponse converti en chaîne de caractères JSON.
    res.end(JSON.stringify(responseData));
});
// Numéro de port sur lequel le serveur écoutera les connexions entrantes.
const port = 7070;
// Démarrage du serveur en écoutant les connexions sur le port spécifié.
server.listen(port, function() {
    // Affichage d'un message indiquant que le serveur a été démarré avec succès.
    console.log('Serveur Ajax au port : ' + port);
    // Affichage de l'URL complète à laquelle le serveur est accessible.
    console.log('http://localhost:' + port);
});
```

- Lancer le serveur avec la commande **node server.js**

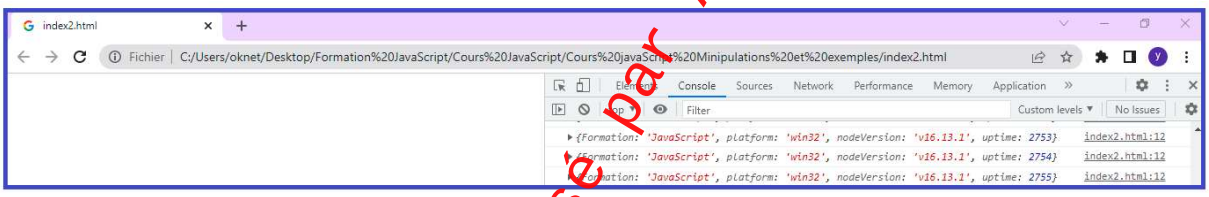
```
C:\Users\oknet\Desktop\Formation JavaScript\Cours JavaScript\Cours javaScript Minipulations et exemples>node server.js
Serveur Ajax au port : 7070
http://localhost:7070
```

- Tester http://localhost:7070/ au navigateur pour voir le résultat

- Coté client au fichier index2.html

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  <script>
    // Définition de la fonction pour rafraîchir les données
    function refresh() {
      // Création d'une nouvelle requête XMLHttpRequest
      const req = new XMLHttpRequest();
      // Ajout événement pour charger les données lorsque la requête est terminée
      req.addEventListener('load', function(evt) {
        // Convertir les chaines JSON reçues en objet JSON
        const data = JSON.parse(this.responseText);
        // Affichage des données dans la console du navigateur
        console.log(data);
      });
      // Ouverture d'une requête GET vers l'URL spécifiée (localhost:7070)
      req.open('GET', 'http://localhost:7070', true);
      // Envoi de la requête
      req.send();
    }
    // Appel périodique de la fonction refresh toutes les 1000 millisecondes
    setInterval(refresh, 1000);
  </script>
</body>
</html>
```

- Tester le fichier index2.html au navigateur pour voir le résultat



La Bibliothèque jQuery

C'est une Librairie JavaScript développée depuis 2006

Parmi ses avantages :

- interface simple et puissante pour écrire du code
- gère les différences entre les navigateurs
- beaucoup de ressources sont disponibles

Pour utiliser **jQuery** :

- installer **jQuery** avec la commande suivante au terminal :
  npm install jquery
- intégrer l'élément script avec le fichier **jquery.js** se trouvant au chemin : **node_modules\jquery\dist\jquery.js**

```
<script src="C:\Users\oknet\node_modules\jquery\dist\jquery.js"></script>
```

- Ou utiliser un CDN (Content Delivery Network) en insérant l'élément script suivant dans la page html :

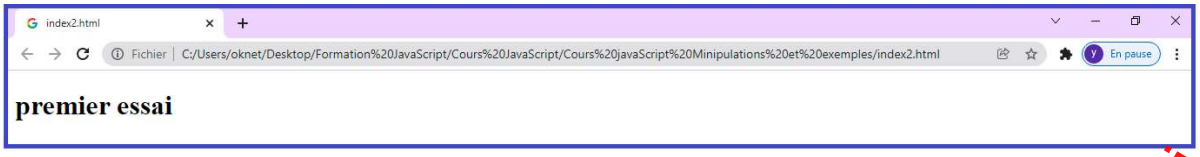
```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
```

Sélectionner avec jQuery :

L'instruction **$(selecteur)** retourne le ou les éléments sélectionnés.

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
 </head>
 <body>
  <h1 id="test"></h1>
  <script>
      $("#test").text("premier essai");
  </script>
 </body>
</html>
```

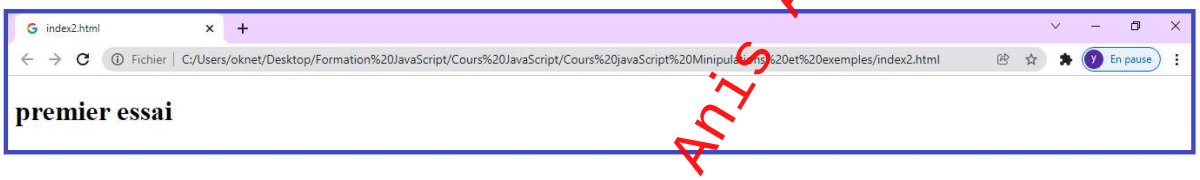
- Tester au navigateur pour voir le résultat



- L'instruction **$(document).ready()** permet d'exécuter du code lorsque le document DOM est totalement chargée.

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
 </head>
 <body>
  <h1 id="test"></h1>
  <script>
     $(document).ready(function() {
                                    $("#test").text("premier essai");
                                  });
  </script>
 </body>
</html>
```

- Tester au navigateur pour voir le résultat



- On utilise une fonction anonyme dans **ready()** (index2.html) :

```
<!DOCTYPE html>
<html>
<head>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
  <ul id="menu">
    <li class="item">
      <p>Ici c'est un élément paragraphe</p>
    </li>
    <li class="item">Ici pas d'élément paragraphe</li>
    <li class="item">Ici pas d'élément paragraphe</li>
  </ul>
  <p>Email: <input type="email" id="email" /></p>
  <p>Text: <input type="text" id="text" /></p>
 </body>
</html>
```

- Dans le fichier script.js

```
$(document).ready(function()
                  {
                    $(".item").css("color", "blue");
                    $("#menu .item p").css("color", "red");
                    $("input[type=email]").css("border", "10px solid blue");
                  });
```

- Tester au navigateur pour voir le résultat

- **Créer des éléments avec jQuery :**

L'instruction **$(element)** crée un nouvel élément.

- Définir le code suivant dans le fichier index2.html

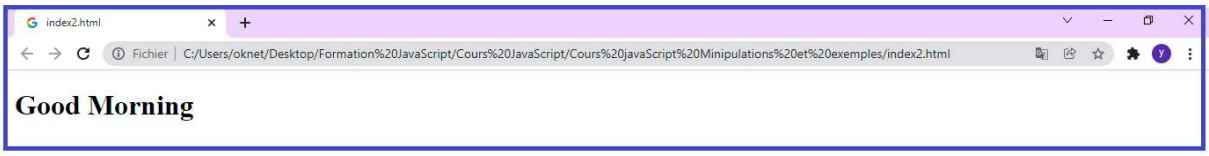
```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
  <div id="div1"></div>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
      let time = new Date().getHours();
      let elem = $("<h1>").attr("id","titre1").hide();
      if (time < 12)
         elem.text("Good Morning");
      else
         elem.text("Good Afternoon");

    $("#div1").append(elem);
    $("#titre1").show("slow");
    });
```

- Tester au navigateur pour voir le résultat



Remarque :

Il est aussi possible d'écrire le code suivant dans le fichier script.js :

```
$(document).ready(function(){
      let time = new Date().getHours();
      if (time < 12)
         elem = $("<h1 id='titre1'>Good Morning</h1>").hide();
      else
         elem = $("<h1 id='titre1'>Good Afternoon</h1>").hide();

      $("#div1").append(elem);
      $("#titre1").show("slow");
    });
```

- **La méthode attr()** pour attribuer et lire les attributs d'un l'élément :

  Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
   <ul>
      <li id="contact">
         <a>Toto</a>
      </li>
   </ul>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
  // Au chargement du document (DOM ready), exécute les instructions suivantes:
  // Change les attributs de l'élément <a> sous l'élément avec l'ID "contact"
  $("#contact a").attr({
      "href" : "http://www.toto.com/",     // Modifie l'URL du lien
      "title" : "Visit Toto website",      // Modifie le titre du lien
      "id" : "atoto"                       // Ajoute un nouvel ID au lien
  });
  // Crée un élément <li> avec le texte du titre de l'élément <a> modifié ci-dessus
  let li = $("<li>").text($("#contact a").attr("title"));
  // Ajoute l'élément <li> créé précédemment à la fin de tous les éléments <ul>
  $("ul").append(li);
});
```

- Tester au navigateur pour voir le résultat

- Nous obtenons le html de la page générée :

```
<!DOCTYPE html>
<html>
  <head>
    <script src="C:\Users\oknet\node_modules\jquery\dist\jquery.js"></script>
    <script src="script.js" type="text/javascript" defer></script>
  </head>
  <body>
    <ul>
      <li id="contact">
        <a href="http://www.toto.com/" title="Visit Toto website" id="atoto">Toto</a>
      </li>
      <li>Visit Toto website</li>
    </ul>
  </body>
</html>
```

- **La méthode css()** pour attribuer et lire les propriétés CSS d'un élément **:**

  Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
   <h1>HTML et CSS</h1>
   <p>
     <strong>La propriété border :</strong>
     <span id="result"></span>
   </p>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
    $("h1").css({
                 "font-size" : "200%",
                 "color" : "#ffffff",
                 "height" : "100px",
                 "width" : "500px",
                 "background-color" : "#61b7ff",
                 "border" : "10px solid #003366"
                });
    $("#result").text($("h1").css("border"));
 });
```

- Tester au navigateur pour voir le résultat

- La méthode **html()** pour attribuer du contenu html à un élément:

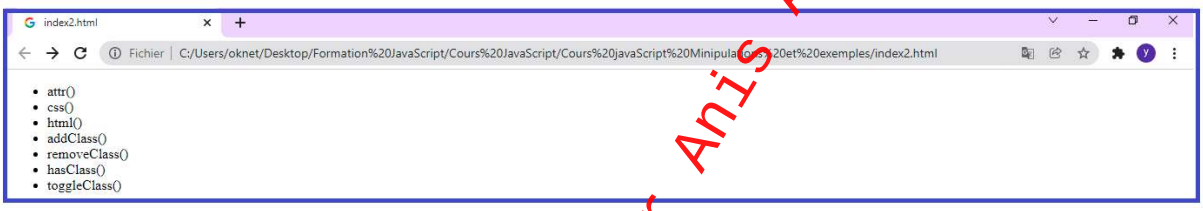
  Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
   <ul id="methods"></ul>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
 let methods = ["attr()", "css()", "html()", "addClass()", "removeClass()", "hasClass()", "toggleClass()"];
 let list="";
 for (let i=0; i < methods.length; i++)
       list += "<li>" + methods[i] +"</li>";
$("#methods").html(list);
});
```

- Tester au navigateur pour voir le résultat



- La méthode **toggleClass()** permet de modifier dynamiquement la valeur de l'attribut class **:**

  Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <link href="styles.css" rel="stylesheet">
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
   <ul>
     <li id="home" class="item">Home</li>
     <li id="about" class="item">About</li>
     <li id="contact" class="item">Contact</li>
   </ul>
 </body>
</html>
```

- Définir le code suivant dans le fichier styles.css

```
.selected {
            background : red;
            color : #fff;
}
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
    // Attends que le DOM soit complètement chargé
    // la méthode .on() pour attacher un gestionnaire d'événement
    $("#home").on("click", function(){
        // La méthode .toggleClass() ajoute ou supprime la classe "selected"
        // Si la classe est déjà présente, elle est supprimée.
        // Si elle est absente, elle est ajoutée.
        $("#home").toggleClass("selected");
    });
});
```

- Tester au navigateur pour voir le résultat

- La méthode **show()** rend visible l'élément et **toggle()** change son display :

  Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
   <ul>
     <li id="slow" hidden>Slow</li>
     <li id="fast" hidden>Fast</li>
     <li id="ms" hidden>Ms</li>
     <li id="toggle">Toggle</li>
     <li id="toggled" hidden>Toggled</li>
   </ul>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
                                $("#slow").show("slow");
                                $("#fast").show("fast");
                                $("#ms").show(1500);
                                $("#toggle").on("click", function(){
                                                                $("#toggled").toggle();
                                                                });
                                });
```

- Tester au navigateur pour voir le résultat

- La méthode **slideToggle()** pour un effet glissant :

  Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
  <dl>
    <dt> Menu 1 </dt>
    <dd>
      <ul>
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
      </ul>
    </dd>
    <dt> Menu 2 </dt>
    <dd>
      <ul>
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
      </ul>
    </dd>
  </dl>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
    $("dt").on("click", function(){
        // Utilise la méthode .next() pour sélectionner l'élément immédiatement
        // suivant l'élément <dt> cliqué.
        // Ensuite, utilise .slideToggle() pour faire glisser (ou masquer)
        // l'élément sélectionné.
        $(this).next().slideToggle();
    });
});
```

- Tester au navigateur en cliquant sur menu 1 ou menu 2 pour voir le résultat

- **Gestion des évènements :**

Voilà quelques méthodes importantes :

- **on()** pour enregistrer un écouteur
- **off()** pour supprimer un écouteur
- **trigger()** pour déclencher un évènement

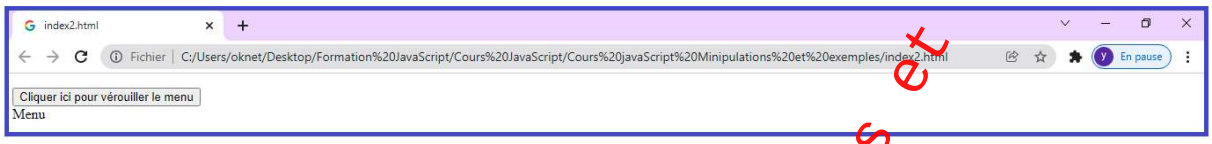
Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
  <dl>
    <button>Cliquer ici pour verrouiller le menu</button>
    <dt> Menu  </dt>
    <dd>
      <ul>
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
      </ul>
    </dd>
  </dl>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).ready(function(){
    // Fonction de bascule pour masquer ou afficher la liste
    function toggler(){
        $(this).next().slideToggle(); // Cacher ou afficher l'élément suivant
    }
    // Attache le gestionnaire d'événement "click" à tous les éléments <dt>
    $("dt").on("click", toggler);
    // Attache un gestionnaire d'événement "click" au bouton
    $("button").on("click", function(){
        // Désactiver du gestionnaire d'événement "click" toggler
        $("dt").trigger("click").off("click", toggler);
    });
});
```

- Tester au navigateur pour voir le résultat



- La méthode **on()** à trois arguments :

Le second paramètre désigne le type d'éléments qui est concerné par l'écouteur (même si l'élément de ce type sera créé plus tard).

L'exemple suivant écoute chaque évènement **click** réalisé sur un élément **<a>.**

Définir le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
 <head>
   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
   <script src="script.js" type="text/javascript" defer></script>
 </head>
 <body>
  <p id="p1"> voila 3 éléments a : </p>
   <a>Un élément a</a><br>
   <a>Un élément a</a><br>
   <a>Un élément a</a><br>
 </body>
</html>
```

- Définir le code suivant dans le fichier script.js

```
$(document).on( "click", "a", function(){
    $("p").css({
        "font-size" : "200%",
        "color" : "#ffffff",
        "height" : "100px",
        "width" : "500px",
        "background-color" : "#61b7ff",
        "border" : "10px solid #003366"
       });
    });
```

- Tester au navigateur pour voir le résultat

Comportement asynchrone

Le comportement asynchrone en JavaScript est crucial pour exécuter des tâches sans bloquer l'exécution du reste du code. Cela est essentiel pour des opérations gourmandes en temps comme les appels réseau, les lectures/écritures de fichiers et autres. Les mécanismes clés pour gérer l'asynchronisme sont **les callbacks**, **les promises** et **l'async/await**.

- **La fonction setTimeout :**

Elle permet d'exécuter une fonction donnée après un laps de temps donné. Il faut bien comprendre l'aspect asynchrone : l'exécution du thread principal ne se place pas en attente.

```
console.log('avant'); // 1
setTimeout(function() {
                        console.log('après 1 mn');  // 3
                       }, 60*1000);
console.log('après'); // 2
/* avant
   après
   après 1 mn */
```

- Tester au terminal pour voir le résultat :

```
PS C:\Users\oknet\Desktop\Exemple> node script.js
avant
après
après 1 mn
PS C:\Users\oknet\Desktop\Exemple>
```

- **Les fonctions setInterval et clearInterval :**
  - La fonction **setInterval** exécute indéfiniment la méthode donnée à un intervalle de temps donné.
  - La fonction **clearInterval** permet de stopper la fonction **setInterval**

```
// Démarre l'affichage de l'heure chaque seconde (1000 millisecondes)
const intervalId = setInterval(function() {
    const currentDate = new Date();
    console.log(`Heure actuelle : ${currentDate.toLocaleTimeString()}`);
}, 1000);

// Arrête l'affichage de l'heure après 5 secondes (5000 millisecondes)
setTimeout(function() {
    clearInterval(intervalId);
    console.log("Affichage de l'heure arrêté.");
}, 5000);
```

- **Callback Functions (Fonctions de Rappel)**

Les fonctions de rappel sont utilisées pour gérer les opérations asynchrones en JavaScript. Une fonction de rappel est passée en argument à une autre fonction, et elle est exécutée lorsque l'opération asynchrone est terminée.

Exemple :

```
// Affiche "Début" dans la console.
console.log("Début");
function asyncOperation(callback) {
  setTimeout(function() {
    // Affiche "Opération asynchrone terminée !" après environ 2 secondes.
    console.log("Opération asynchrone terminée !");
    // Appelle la fonction de rappel fournie en argument.
    callback();
  }, 2000);
// Attend 2 secondes avant d'exécuter le code à l'intérieur de la fonction de rappel.
}

asyncOperation(function() {
// Affiche "Fonction de rappel exécutée !" quand la fonction de rappel est exécutée.
  console.log("Fonction de rappel exécutée !");
});
// Affiche "Suite" dans la console.
console.log("Suite");
```

- Tester au terminal pour voir le résultat :

```
PS C:\Users\oknet\Desktop\Exemple> node script.js
Début
Suite
Opération asynchrone terminée !
Fonction de rappel exécutée !
```

- **Promises (Promesses)**

Les promesses sont une manière structurée de gérer l'asynchronisme. Une promesse représente une valeur future d'une opération asynchrone et peut être dans l'état "en attente", "résolue" ou "rejetée". Les méthodes **.resolve**, **.reject**, **.then** et **.catch** sont utilisées pour travailler avec les promesses.

Exemple :

```
// Affiche "Début" dans la console
console.log("Début");
// Crée une promesse pour une opération asynchrone
const promise = new Promise((resolve, reject) => {
  // Simule une attente de 2 secondes
  setTimeout(() => {
    // Génère un nombre aléatoire entre 0 et 1
    const randomNumber = Math.random();
    if (randomNumber > 0.5) {
      // Si le nombre est grand, réussit avec la valeur
      resolve(randomNumber);
    }
    else
    {
      // Sinon, échoue avec une erreur
      reject(new Error("Nombre trop petit."));
    }
  }, 2000);
});

// Utilise ".then()" pour le succès et ".catch()" pour l'échec
promise
  .then((result) => {
    // Affiche la valeur si réussi
    console.log("Réussi avec :", result);
  })
  .catch((error) => {
    // Affiche le message d'erreur si échoué
    console.error("Échoué avec :", error.message);
  });
// Affiche "Suite" dans la console
console.log("Suite");
```

- Tester au terminal pour voir le résultat :

```
PS C:\Users\oknet\Desktop\Exemple> node script.js
Début
Suite
Réussi avec : 0.6698004659544012
PS C:\Users\oknet\Desktop\Exemple>
```

- **Async/Await**

L'async/await est une syntaxe moderne qui rend le code asynchrone plus lisible et similaire au code synchrone. Une fonction marquée comme async renvoie toujours une promesse, et await est utilisé pour attendre la résolution d'une promesse.

Exemple :

```
console.log("Début");
// Définit une fonction asynchrone
async function asyncFunction() {
  try {
    // Crée une promesse pour une opération asynchrone
    const result = await new Promise((resolve, reject) => {
      setTimeout(() => {
        // Génère un nombre aléatoire entre 0 et 1
        const randomNumber = Math.random();
        if (randomNumber > 0.5)
        {
          resolve(randomNumber); // Réussit la promesse avec la valeur
        }
        else
        {
          // Échoue la promesse avec une erreur
          reject(new Error("La valeur est trop basse."));
        }
      }, 2000);
    });

    // Affiche la valeur résolue dans la console
    console.log("Résolu avec la valeur :", result);
  }
  catch (error)
  {
    // Affiche le message d'erreur si la promesse est rejetée
    console.error("Rejeté avec l'erreur :", error.message);
  }
}

// Appelle la fonction asynchrone
asyncFunction();
// Affiche "Suite" dans la console.
console.log("Suite");
```

- Tester au terminal pour voir le résultat :

```
PS C:\Users\oknet\Desktop\Exemple> node script.js
Début
Suite
Résolu avec la valeur : 0.7895239421464846
```

- **Gestion des Erreurs**

La gestion des erreurs est cruciale lors de la manipulation d'opérations asynchrones. Vous pouvez utiliser **try**...**catch** pour capturer et gérer les erreurs synchrones, tandis que **.catch** est utilisé pour attraper les erreurs dans les promesses.

Exemple :

```
console.log("Début");
// Définit une fonction asynchrone
async function asyncFunction() {
  try {
    // Crée une promesse pour une opération asynchrone
    const result = await new Promise((resolve, reject) => {
      setTimeout(() => {
        // Génère un nombre aléatoire entre 0 et 1
        const randomNumber = Math.random();
        if (randomNumber > 0.5)
        {
          // Réussit la promesse avec la valeur
          resolve(randomNumber);
        }
        else
        {
          // Échoue la promesse avec une erreur
          reject(new Error("La valeur est trop basse."));
        }
      }, 2000);
    });
    // Affiche la valeur résolue dans la console
    console.log("Résolu avec la valeur :", result);
  }
  catch (error)
  {
    // Affiche le message d'erreur si la promesse est rejetée
    console.error("Rejeté avec l'erreur :", error.message);
  }
}

// Appelle la fonction asynchrone
asyncFunction();
// Affiche "Suite" dans la console
console.log("Suite");
```

- Tester au terminal pour voir le résultat :

```
PS C:\Users\oknet\Desktop\Exemple> node script.js
Début
Suite
Résolu avec la valeur : 0.6770589484317693
```

- **Exercice : Lecture Asynchrone de Fichier avec Promesses**

Pratiquer la lecture asynchrone d'un fichier en utilisant les Promesses en JavaScript.

Utiliser le module fs (système de fichiers) pour effectuer la lecture.

```
// Importation du module fs (système de fichiers) pour gérer les opérations de
lecture/écriture de fichiers
const fs = require('fs');
// Création d'une fonction qui renvoie une promesse pour la lecture du fichier
const readFilePromise = (fileName) => {
  return new Promise((resolve, reject) => {
    // Utilisation de fs.readFile pour lire le fichier
    fs.readFile(fileName, (err, data) => {
      if (err) {
        // Si une erreur se produit, rejeter la promesse avec un objet d'erreur
        reject(new Error('Error : ' + err.message));
      } else {
        // Si la lecture réussit, résoudre la promesse avec le contenu du fichier
        resolve(String(data));
      }
    });
  });
};

const fileName = 'toto.txt';
// Utilisation de la fonction readFilePromise pour lire le fichier de manière
asynchrone
readFilePromise(fileName)
  .then((fileContent) => {
    // Gestion du succès : affichage du contenu du fichier
    console.log(fileContent);
  })
  .catch((error) => {
    // Gestion des erreurs : affichage du message d'erreur
    console.error(error.message);
  });
```


Conclusion

Le comportement asynchrone en JavaScript est essentiel pour créer des applications performantes et réactives. En utilisant les callbacks, les promesses, l'async/await et la gestion des erreurs, on peut gérer efficacement les opérations asynchrones tout en maintenant un code propre et lisible. Cela permet de construire des applications qui répondent rapidement aux utilisateurs, même lors de l'exécution d'opérations gourmandes en temps.

Les cookies

Les cookies sont de petits fichiers texte stockés dans le navigateur d'un utilisateur. Ils sont utilisés pour stocker des informations entre les visites d'un site. Voici comment les utiliser en JavaScript.

- **Créer un Cookie :**

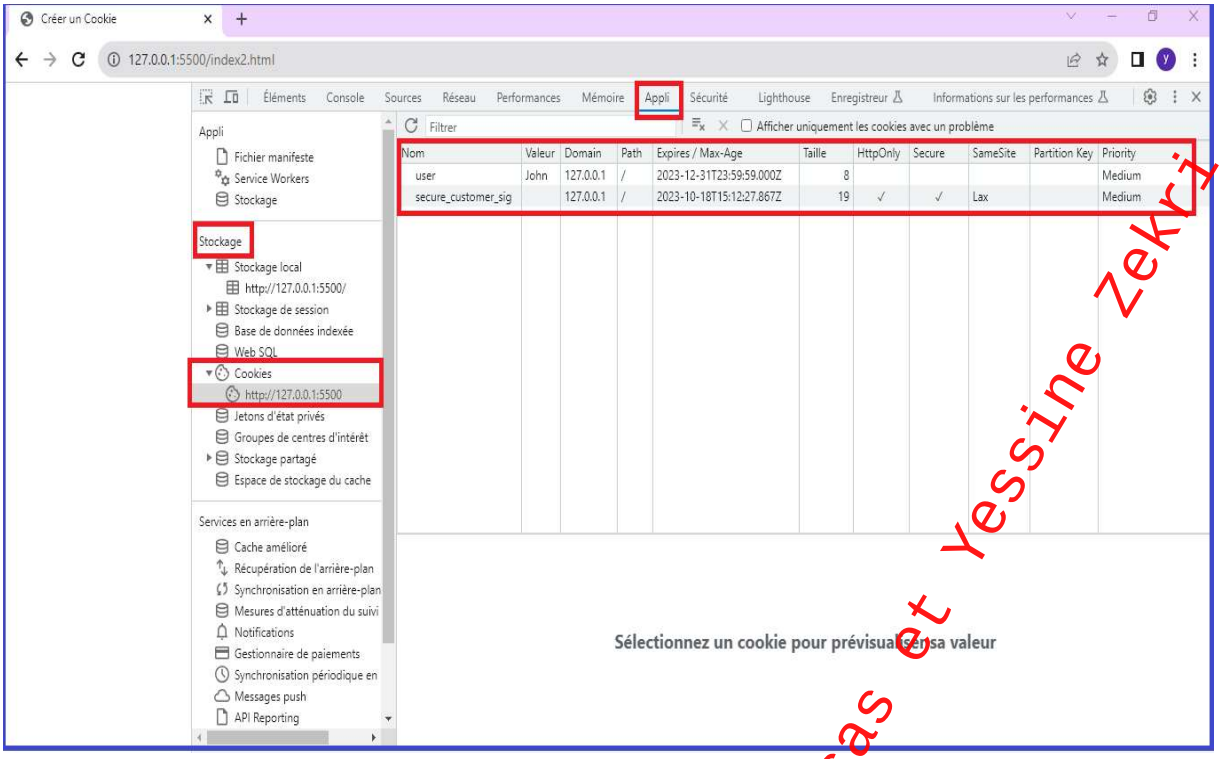
  Créez un cookie avec un nom, une valeur et une durée de vie (fichier index2.html):

```
<!DOCTYPE html>
<html>
<head>
  <title>Créer un Cookie</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Crée un cookie avec le nom "user" et la valeur "John"
document.cookie = "user=John; expires=Thu, 31 Dec 2023 23:59:59 UTC; path=/;";
console.log("Cookie créé !");
```

- Tester au navigateur pour voir le resultat :



- **Lire le contenu d'un Cookie :**

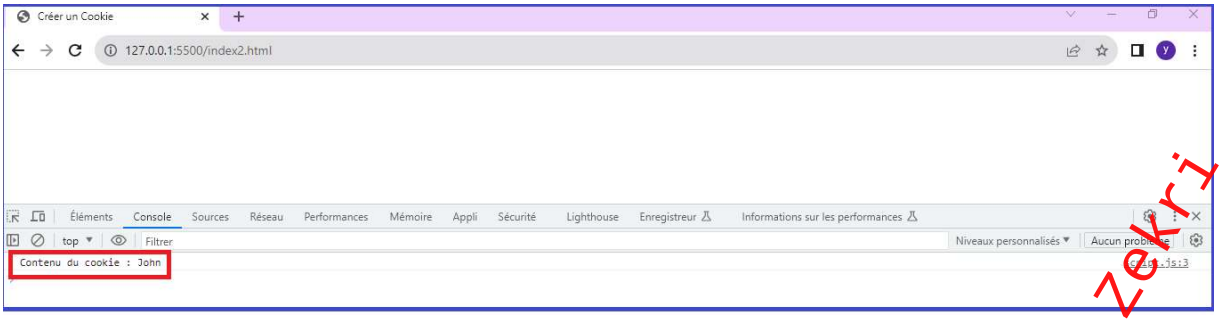
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Lire un Cookie</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

  Ajouter le code suivant au fichier script.js :

```
// Lit le contenu du cookie "user"
const cookieValue = document.cookie.split('=')[1];
console.log("Contenu du cookie :", cookieValue);
```

- Tester à la console pour voir le résultat :



- **Modifier un cookie :**

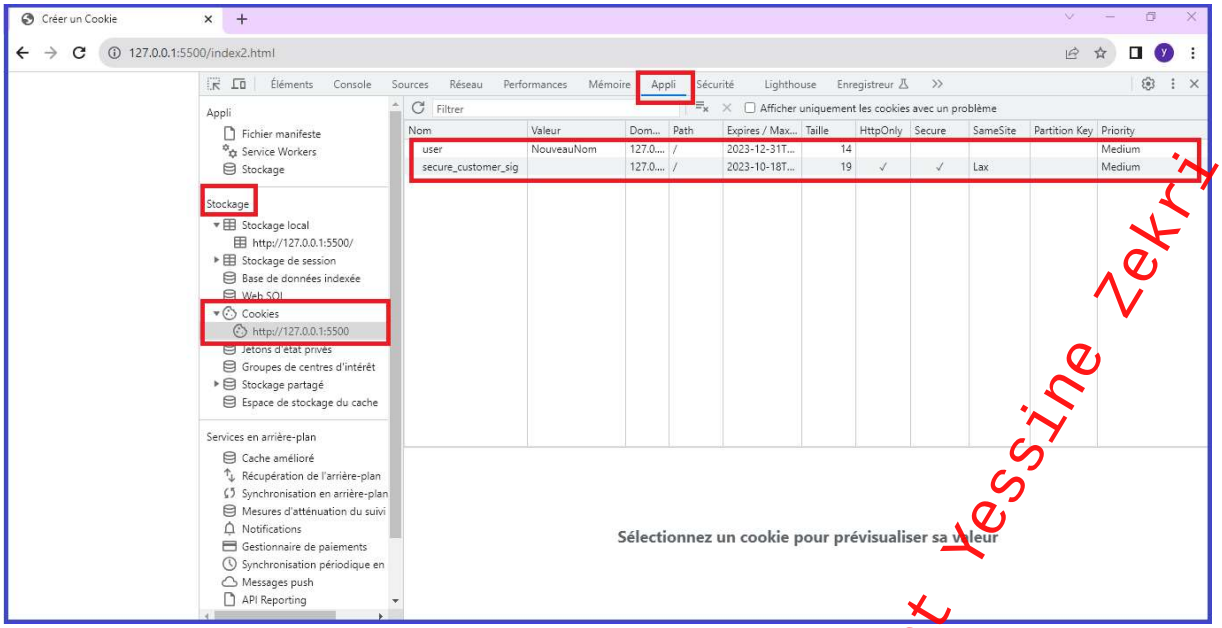
Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Modifier un Cookie</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier scriptjs :

```
// Modifie la valeur du cookie "user"
document.cookie = "user=NouveauNom; expires=Thu, 31 Dec 2023 23:59:59 UTC; path=/;";
console.log("Cookie modifié !");
```

- Tester au navigateur pour voir le resultat :



- **Supprimer un Cookie en définissant sa date d'expiration dans le passé:**

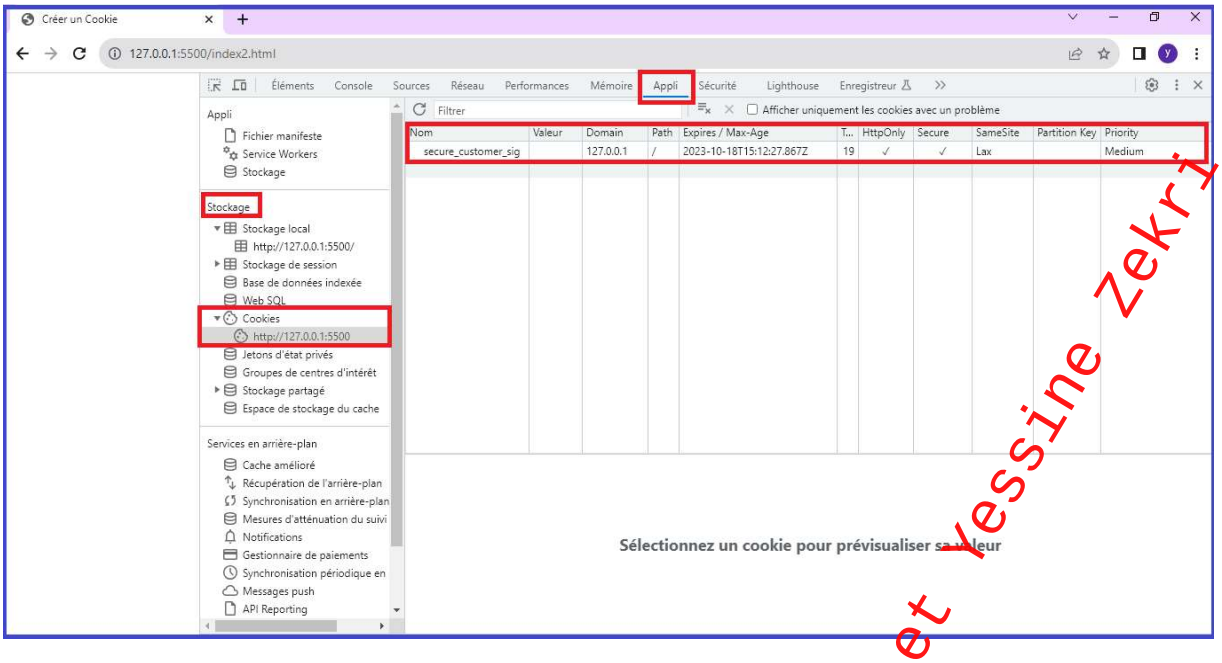
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Supprimer un Cookie</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

  Copier le code suivant dans le fichier script.js :

```
// Supprime le cookie "user"
document.cookie = "user=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
console.log("Cookie supprimé !");
```

- Tester au navigateur pour voir le resultat :



Conclusion: Les cookies ont des limitations, comme leur taille limitée et des problèmes de sécurité potentiels. Ils ne devraient pas être utilisés pour stocker des données sensibles. Des alternatives plus modernes et sûres, comme le stockage local (localStorage) et de session (sessionStorage), sont souvent préférables.

Le localStorage

Le localStorage est un espace de stockage côté client qui permet de stocker des données en paires clé-valeur. Contrairement aux cookies, il offre une plus grande capacité de stockage et les données ne sont pas envoyées au serveur avec chaque requête. Voici comment l'utiliser en JavaScript.

- **Ajouter un objet User au localStorage :**

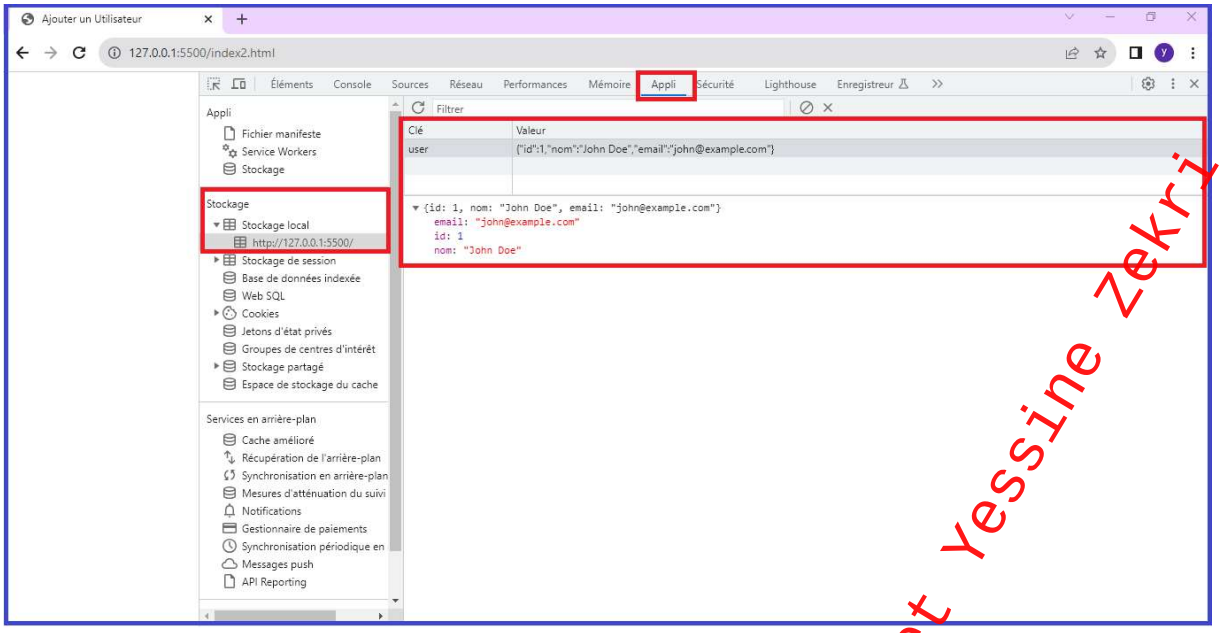
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Ajouter un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

  Copier le code suivant dans le fichier script.js :

```
// Crée un objet utilisateur
const user = {
  id: 1,
  nom: "John Doe",
  email: "john@example.com"
};
// Stocke l'objet utilisateur dans le localStorage
localStorage.setItem("user", JSON.stringify(user));
console.log("Utilisateur ajouté !");
```

- Tester au navigateur pour voir le resultat :



- **Lire un User depuis le localStorage :**

  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Lire un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Récupère l'objet utilisateur depuis le localStorage
const userJSON = localStorage.getItem("user");
const user = JSON.parse(userJSON);
console.log("Utilisateur lu :", user);
```

- Tester au navigateur pour voir le resultat :

- **Modifier les propriétés d'un objet User dans le localStorage :**

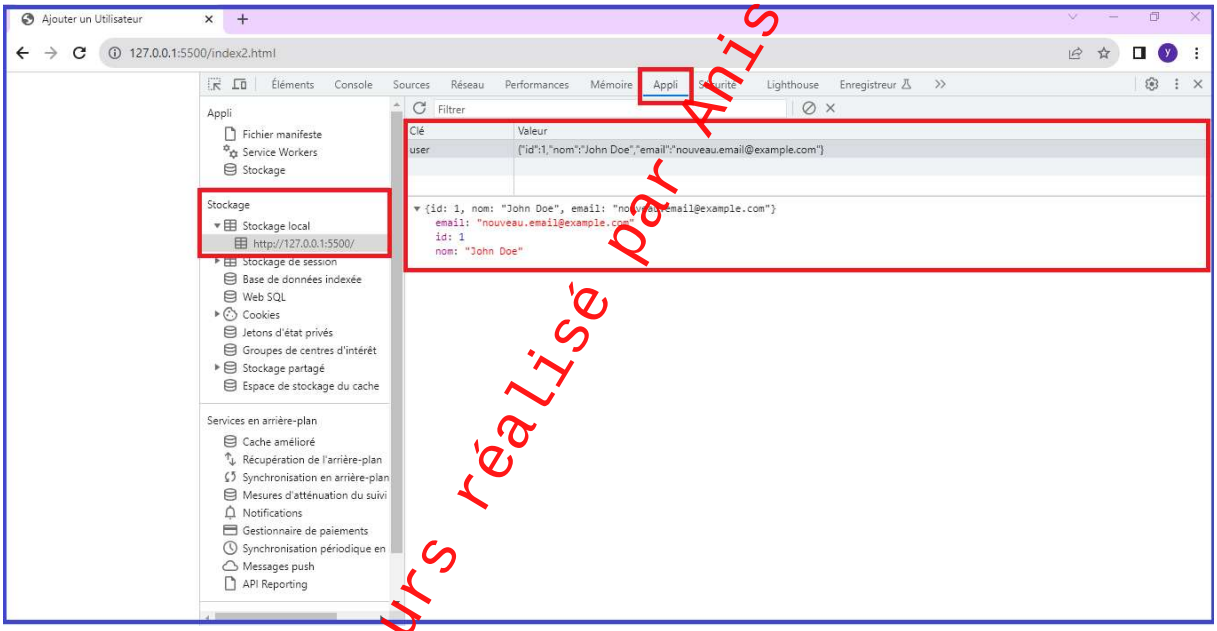
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Modifier un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Récupère l'objet utilisateur depuis le localStorage
const userJSON = localStorage.getItem("user");
const user = JSON.parse(userJSON);
// Modifie la propriété email
user.email = "nouveau.email@example.com";
// Stocke l'objet utilisateur modifié dans le localStorage
localStorage.setItem("user", JSON.stringify(user));
console.log("Utilisateur modifié !");
```

- Tester au navigateur pour voir le resultat :



- **Supprimer un objet User du localStorage :**

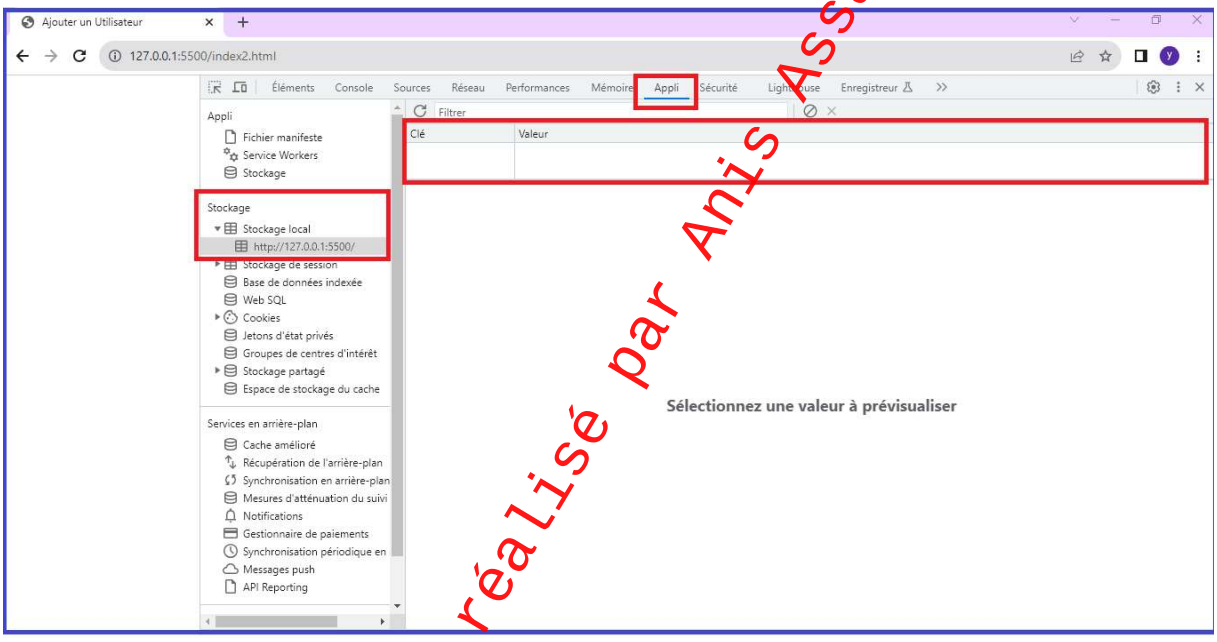
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Supprimer un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Supprime l'objet utilisateur du localStorage
localStorage.removeItem("user");
console.log("Utilisateur supprimé !");
```

- Tester au navigateur pour voir le resultat :



Conclusion : Le localStorage est une excellente solution pour stocker des données côté client de manière simple et efficace. Cependant, il convient de noter que les données du localStorage sont stockées en tant que chaînes de caractères, donc si vous avez besoin de stocker des objets JavaScript, vous devez les sérialiser et les désérialiser à l'aide de **JSON.stringify** et **JSON.parse**.

Le sessionStorage

Le sessionStorage est un espace de stockage côté client similaire au localStorage, mais avec une durée de vie limitée à la durée de la session en cours. Il est utile pour stocker temporairement des données pendant la navigation sur une seule page ouverte. Voici comment l'utiliser en JavaScript.

- **Ajouter un objet User au sessionStorage :**

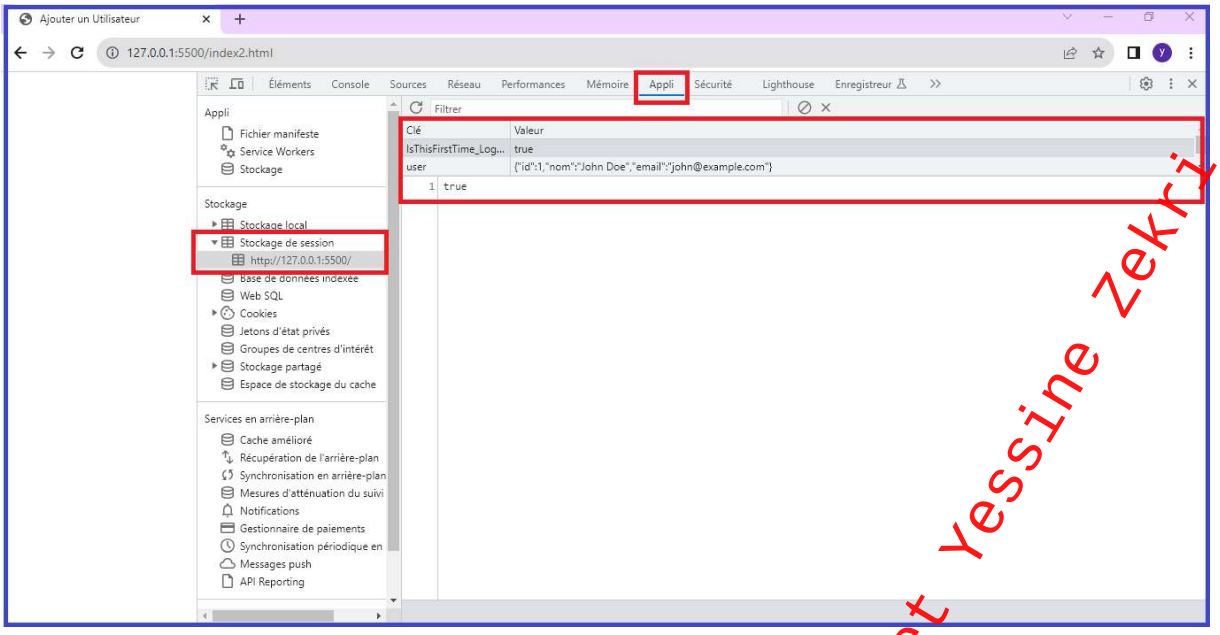
  Copier le code suivant dans le fichier index2.html :

```html
<!DOCTYPE html>
<html>
<head>
  <title>Ajouter un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```javascript
// Crée un objet utilisateur
const user = {
  id: 1,
  nom: "John Doe",
  email: "john@example.com"
};
// Stocke l'objet utilisateur dans le sessionStorage
sessionStorage.setItem("user", JSON.stringify(user));
console.log("Utilisateur ajouté !");
```

- Tester au navigateur pour voir le resultat :



- **Lire un objet User depuis le sessionStorage :**

  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Lire un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Récupère l'objet utilisateur depuis le sessionStorage
const userJSON = sessionStorage.getItem("user");
const user = JSON.parse(userJSON);
console.log("Utilisateur lu :", user);
```

- Tester au navigateur pour voir le resultat :

- **Modifier les propriétés d'un objet User dans le sessionStorage :**

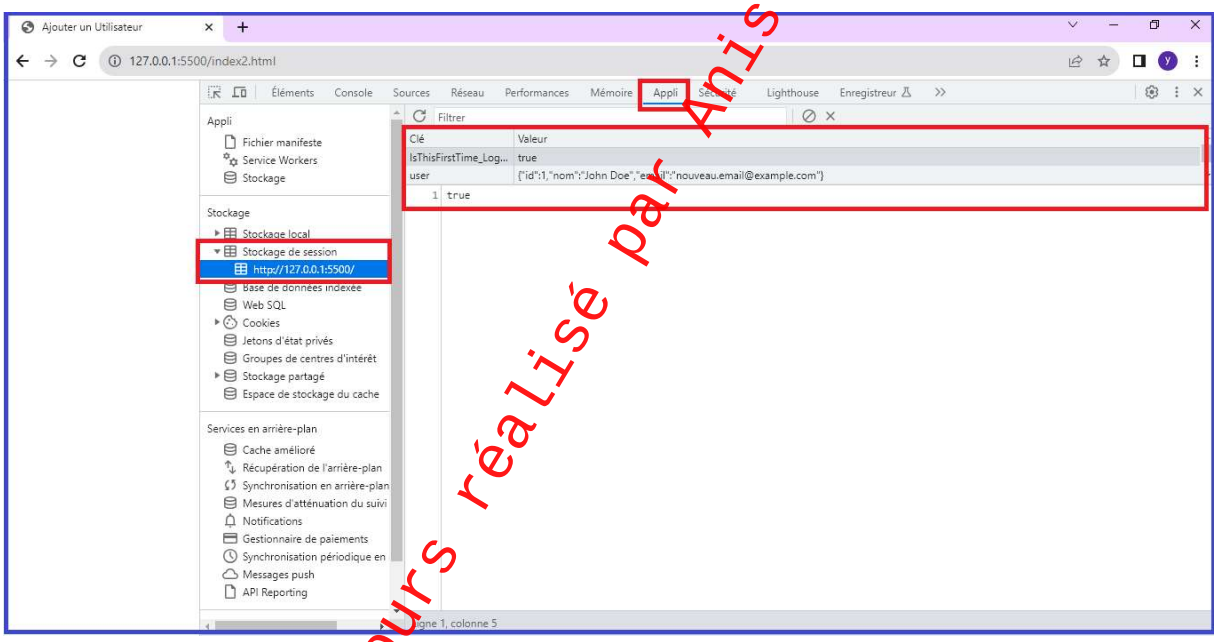
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Modifier un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Récupère l'objet utilisateur depuis le sessionStorage
const userJSON = sessionStorage.getItem("user");
const user = JSON.parse(userJSON);
// Modifie la propriété email
user.email = "nouveau.email@example.com";
// Stocke l'objet utilisateur modifié dans le sessionStorage
sessionStorage.setItem("user", JSON.stringify(user));
console.log("Utilisateur modifié !");
```

- Tester au navigateur pour voir le resultat :



- **Supprimer un objet User du sessionStorage :**

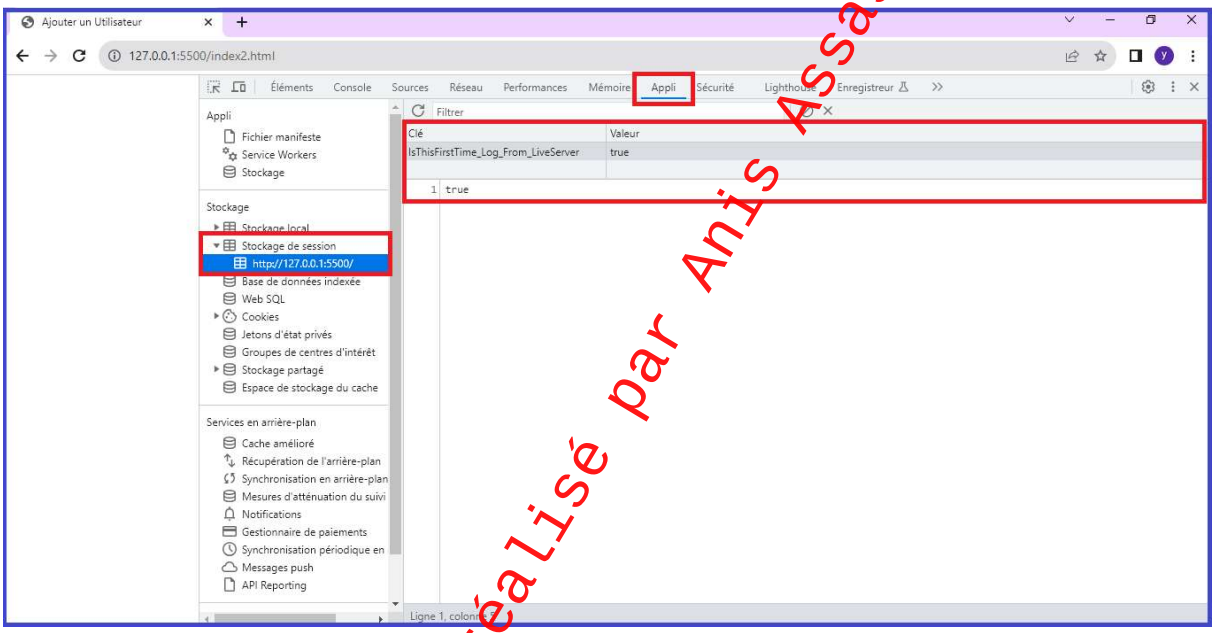
  Copier le code suivant dans le fichier index2.html :

```
<!DOCTYPE html>
<html>
<head>
  <title>Supprimer un User</title>
  <script src="script.js" type="text/javascript" defer></script>
</head>
<body>
</body>
</html>
```

Copier le code suivant dans le fichier script.js :

```
// Supprime l'objet utilisateur du sessionStorage
sessionStorage.removeItem("user");
console.log("Utilisateur supprimé !");
```

- Tester au navigateur pour voir le resultat :



Conclusion : Le sessionStorage est particulièrement utile pour stocker des données qui doivent être disponibles pendant la session en cours. Comme pour le localStorage, il faut s'assurer de sérialiser et désérialiser les objets JavaScript en utilisant **JSON.stringify** et **JSON.parse** pour stocker et récupérer des données.