



# LE VOCABULAIRE DU WEB

# QUELQUES DÉFINITIONS

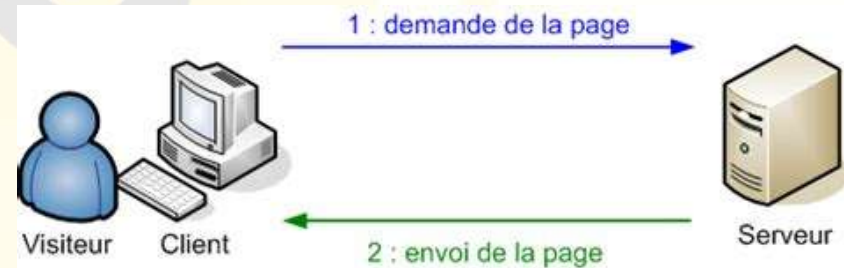
## URL Uniform Resource Locator

- Format de nommage universel pour désigner une ressource sur Internet.
  - Protocole (HTTP, HTTPS, FTP...)
  - Nom de domaine ou l'adresse IP de l'ordinateur hébergeant la ressource.
  - Port logiciel utilisé par le protocole pour la mise en place de connexion (HTTP = 80 port par défaut)
  - La ressource demandée.



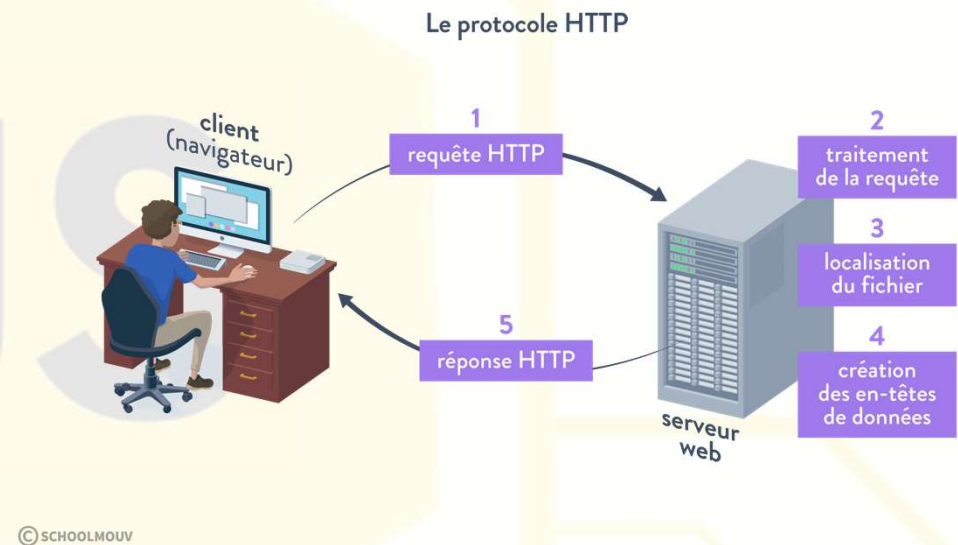
## Requête HTTP HyperText Transfer Protocol

- Transfert de messages avec des en-têtes en codage MIME



# PROTOCOLE HTTP

1. Le navigateur effectue une requête HTTP.
  - Une requête est composée :
    - D'un en-tête et du corps de la requête
  - Différentes méthodes :
    - GET, POST
    - HEAD, PUT, DELETE
2. Le serveur traite la requête puis envoie une réponse HTTP.
  - Une Réponse est composée :
    - D'un statut :
      - protocole utilisé
      - code de retour et de son libellé
  - D'un en-tête,
    - Du corps de la réponse



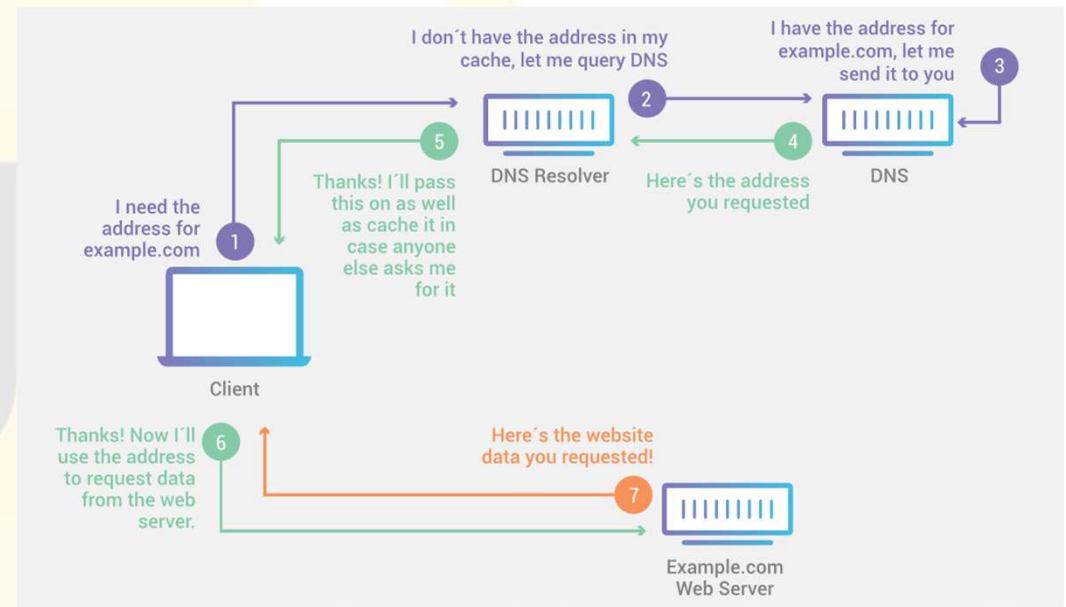
# DOMAIN NAME SYSTEM

**DNS** : le DNS permet de traduire les noms de domaine en une adresse IP correspondant à une machine (serveur).

- Chaque appareil connecté à Internet dispose d'une adresse IP unique que les autres appareils utilisent afin de le trouver.

Grâce aux serveurs DNS, les internautes n'ont pas à mémoriser les adresses IP, ni les adresses IP alphanumériques plus récentes et plus complexes.

Ceci grâce aux fichiers de zone placés sur les serveurs DNS





# REQUÊTES HTTP

## COMMUNIQUER AVEC UN SERVEUR

# QUELQUES NOTIONS

Quelques notions théoriques qu'il faut connaître avant de se lancer dans la communication avec un serveur en JavaScript. □□□□□□□

**Service Web** : programme s'exécutant sur un serveur accessible depuis internet et fournissant un service.

- Pour ce faire, il met à disposition une API.

**API** : Application Programming Interface - interface de communication avec les services Web, au travers de requêtes.

**Requêtes** : données qui respectent le protocole de communication et qui sont envoyées au serveur.

**Protocoles** : SMTP (envoi de mails), IMAP (réception de mails), HTTP et HTTPS, FTP, WebDAV, etc...

**Requêtes HTTP asynchrones** : **AJAX** (Asynchronous JavaScript and XML) est la technologie utilisée pour gérer ce type de requêtes.

- Cela permet de ne pas bloquer le navigateur pendant l'attente d'une réponse du serveur.

**JSON** : le format de données standard actuel pour l'échange des données sur le WEB.

# LE PROTOCOLE HTTP(S)

HTTP : HyperText Transfert Protocol

- Communiquer avec un site internet, chargement des pages HTML, des styles CSS, etc...
- Envoi et récupération d'informations avec les formulaires.

- Méthodes :
  - **GET** : permet de récupérer des ressources
  - **POST** : permet de créer ou modifier une ressource
  - **PUT** : permet de modifier une ressource
  - **DELETE** : permet de supprimer une ressource
  - **HEAD** : demande des informations sur la ressource sans obtenir la ressource.
  - **TRACE, OPTION, CONNECT**...
- **URL** : l'adresse du service web à atteindre
- **Données** : les données qu'on envoie mais aussi qu'on reçoit
- **Code HTTP** : code numérique qui indique comment s'est déroulée la requête
  - 200 : tout s'est bien passé
  - 404 : la ressource n'existe pas
  - 500 : une erreur avec le service web
  - ... [https://fr.wikipedia.org/wiki/Liste\\_des\\_codes\\_HTTP](https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP)

# GET vs POST

## GET

Cette méthode de requête existe depuis le début du Web. Elle est utilisée pour demander une ressource, par exemple un fichier HTML, au serveur Web.

- Paramètres URL

La requête GET peut recevoir des informations supplémentaires que le serveur Web doit traiter.

Ces paramètres d'URL sont simplement ajoutés à l'URL. La syntaxe est très simple :

- La chaîne de requête est introduite par un ? Et chaque paramètre est nommé, et composé donc d'un nom et d'une valeur : Nom=Valeur
- Si plusieurs paramètres doivent être inclus, ils sont reliés par un &
- un encodage des caractères spéciaux est effectué ( exemple : @).

## POST

Si vous souhaitez envoyer de grandes quantités de données, par exemple des images, ou des données confidentielles de formulaires au serveur, la méthode GET n'est pas idéale car toutes les données que vous envoyez sont écrites ouvertement dans la barre d'adresse du navigateur.

Dans ces cas, la méthode POST est la plus adaptée.

Cette méthode n'écrit pas les paramètres de l'URL, mais les ajoute à l'en-tête HTTP.

- Les requêtes POST sont principalement utilisées pour les formulaires en ligne.





# RÉCUPÉRATION DES DONNÉES DU FORMULAIRE

[HTTPS://DEVELOPER.MOZILLA.ORG/FR/DOCS/WEB/API/  
WINDOW](https://developer.mozilla.org/fr/docs/web/api/window)

# EXEMPLE

Soit le formulaire suivant :

**Formulaire de Saisie**

Nom :

Prénom :

Email :

Date d'anniversaire :

Adresse :

```
<form id="userForm" action="compte.html" method="get">
  <div class="form-group">
    <label for="nom">Nom :</label>
    <input type="text" id="nom" name="nom" required pattern="[A-Za-zÀ-Öø-öÿ\s]+">
    <div class="error" id="nomError"></div>
  </div>
  <div class="form-group">
    <label for="prenom">Prénom :</label>
    <input type="text" id="prenom" name="prenom" required pattern="[A-Za-zÀ-Öø-öÿ\s]+">
    <div class="error" id="prenomError"></div>
  </div>
  <div class="form-group">
    <label for="email">Email :</label>
    <input type="email" id="email" name="email" required>
    <div class="error" id="emailError"></div>
  </div>
  <div class="form-group">
    <label for="dateAnniversaire">Date d'anniversaire :</label>
    <input type="date" id="dateAnniversaire" name="dateAnniversaire" required>
    <div class="error" id="dateAnniversaireError"></div>
  </div>
  <div class="form-group">
    <label for="adresse">Adresse :</label>
    <input type="text" id="adresse" name="adresse" required>
    <div class="error" id="adresseError"></div>
  </div>
  <button type="submit">Soumettre</button>
</form>
```

# WINDOW

Lors de l'envoi du formulaire par la méthode GET, on envoie au travers de la requête HTTP [de manière visible](#), les informations saisies dans notre formulaire.

Avec [window.location](#), on obtient l'ensemble des propriétés disponibles.

- [window](#) représente notre page en cours
- [location](#) contient toutes les informations de la page courante

```
> window.location
< Location {ancestorOrigins: DOMStringList, href: 'http://127.0.0.1:5500/HTTP_et_API/compte.html?nom=...versaire=2025-03-10&adresse=centre+afpa+de+Pompey', origin: 'http://127.0.0.1:5500', protocol: 'http:', host: '127.0.0.1:5500', ...}
  ▶ ancestorOrigins: DOMStringList {length: 0}
  ▶ assign: f assign()
    hash: ""
    host: "127.0.0.1:5500"
    hostname: "127.0.0.1"
    href: "http://127.0.0.1:5500/HTTP_et_API/compte.html"
    origin: "http://127.0.0.1:5500"
    pathname: "/HTTP_et_API/compte.html"
    port: "5500"
    protocol: "http:"
    reload: f reload()
    replace: f replace()
    search: "?nom=boebion&prenom=jerome&email=boebion@boebion.com"
    toString: f toString()
    valueOf: f valueOf()
    Symbol(Symbol.toPrimitive): undefined
    [[Prototype]]: Location
```

## WINDOW.LOCATION.SEARCH

L'attribut `search` contient l'ensemble des paramètres envoyés au serveur sous la forme

- `?param=valeur&param=valeur...`

Ensuite, on peut aisément récupérer cette donnée dans une variable et en extraire les informations dans un tableau.

```
> let param = window.location.search
< undefined
> console.info(param);
? VM469:1
nom=boebion&prenom=jerome&email=jero%40fr
ee.fr&dateAnniversaire=2025-03-
10&adresse=centre+afpa+de+Pompey
```

# RÉCUPÉRATION DES DONNÉES D'UNE URL

Maintenant que nous connaissons les éléments à cibler pour récupérer les données du formulaire, il nous reste à écrire un script JS.

Voici un exemple de récupération des données **de manière basique** au travers d'une URL.

- Les tests dans ce code permettent de remplacer les codes ascii que le formulaire utilise lors de l'envoi.
  - *Dans cet exemple, pour l'email par exemple il remplace : par le code ascii correspondant %40 et les espaces par +*

```
function getParamsURL() {

    // objet contenant les parametres
    let varParams = {};
    // récupération du l'url en retirant le ?
    let search = window.location.search.substring(1);
    // contrôle
    console.info(search);

    // decomposition des parametre en retirant le &
    let varSearch = search.split('&');
    // contrôle
    console.info(varSearch);

    // parcours des éléments et contruction de mon objet
    for (let i=0; i < varSearch.length; i++) {
        // suppression du =
        let parameter = varSearch[i].split('=');
        // mise en forme du caractère ASCII @
        if (parameter[0] === "email") {
            parameter[1] = parameter[1].replace('%40', '@');
        }
        // mise en forme du caractère ASCII +
        if (parameter[0] === "adresse") {
            parameter[1] = parameter[1].replaceAll(' ', ' ');
        }
        // contruction de la réponse
        varParams[parameter[0]] = parameter[1].toUpperCase();
    }
    // controle
    console.info(varParams);

    return varParams;
}
```

# AFFICHAGE

Ensuite dans la page de récupération, là où le formulaire nous redirige, nous n'avons plus qu'à afficher les données saisies.

```
document.addEventListener("DOMContentLoaded", function () {  
  
    // appel de la fonction  
    let params = getParamsURL();  
  
    console.dir(params);  
  
    // Mise en forme dans la page HTML  
    document.getElementById("nom").textContent = params.nom;  
    document.getElementById("prenom").textContent = params.prenom;  
    document.getElementById("email").textContent = params.email;  
    document.getElementById("dateAnniversaire").textContent = params.dateAnniversaire;  
    document.getElementById("adresse").textContent = params.adresse;  
  
});
```





# FORMDATA

[HTTPS://DEVELOPER.MOZILLA.ORG/FR/D  
OCS/WEB/API/FORMDATA](https://developer.mozilla.org/fr/docs/web/api/formdata)

# FORMDATA

L'objet `FormData` a été standardisé et facilite grandement l'envoi vers un serveur.

- Il peut être utilisé indépendamment d'un formulaire, en lui ajoutant une à une les données à transmettre grâce à sa méthode `append`.
- Cette méthode prend en paramètres le nom et la valeur de la donnée ajoutée (clé/valeur).

Ensuite, on peut retrouver les données contenant dans notre objet `FormData` avec la méthode `entries()` qui retourne un `itérateur` permettant d'accéder à l'ensemble des données.

*Cet objet est souvent utilisé avec la méthode POST*

```
document.getElementById('monFormulaire').addEventListener('submit', function(event) {  
    event.preventDefault(); // Empêche l'envoi du formulaire  
  
    // Crée un nouvel objet FormData à partir du formulaire  
    const formData = new FormData(event.target);  
  
    // Affiche les données du formulaire dans la console  
    for (let [name, value] of formData.entries()) {  
        console.log(`${name}: ${value}`);  
    }  
});
```

<https://fr.javascript.info/formdata>



# MÉTHODES DISPONIBLES

Diverses méthodes sont disponibles avec formData

<https://developer.mozilla.org/fr/docs/Web/API/FormData>

- `append()` : Ajoute une nouvelle valeur à une clé existante dans un objet FormData, ou ajoute la clé si elle n'existe pas encore.
- `delete()` : Supprime une paire clé/valeur d'un objet FormData.
- `entries()` : Renvoie un itérateur permettant de passer en revue toutes les paires clé/valeur contenues dans cet objet.
- `get()` : Renvoie la première valeur associée à une clé donnée à partir d'un objet FormData.
- `getAll()` : Renvoie un tableau de toutes les valeurs associées à une clé donnée à partir d'un objet FormData.
- `has()` : Renvoie un booléen indiquant si un objet FormData contient une certaine clé.
- `key()` : Renvoie un itérateur permettant de parcourir toutes les clés des paires clé/valeur contenues dans cet objet.
- `set()` : Renvoie un itérateur permettant de parcourir toutes les valeurs contenues dans cet objet.
- `values()` : Renvoie un itérateur permettant de parcourir toutes les valeurs contenues dans cet objet.



# LOCALSTORAGE ET SESSION STORAGE AU SEIN DU NAVIGATEUR

# LOCALSTORAGE

- fonctionnalités JavaScript qui permet de stocker des données de manière persistante ou temporaire dans le navigateur web.
- Les données stockées dans `localStorage` restent disponibles même après la fermeture et la réouverture du navigateur.
- Les données stockées dans `sessionStorage` restent disponibles dans la session ou l'onglet du navigateur.
- utilisé pour stocker des données locales telles que des préférences utilisateur, des données de session ou des informations de configuration.
- accessibles uniquement via JavaScript et spécifiques au domaine et au protocole utilisés pour accéder à la page.

• **Attention** : En termes de sécurité web, l'utilisation doit être abordée avec prudence, car il peut présenter certains risques si les données sensibles sont mal gérées ou mal protégées.

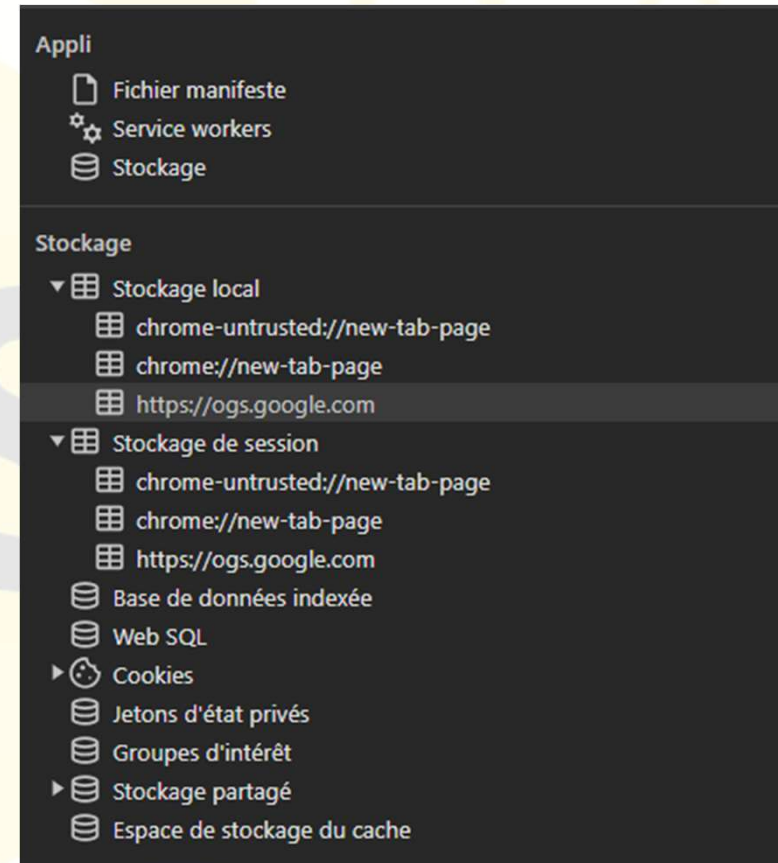
- **Données sensibles** : Ces données pourraient être accessibles à des scripts malveillants ou à d'autres attaques.
- **Injection de code** : Ne stockez pas de données directement à partir d'entrées utilisateur sans validation et échappement appropriés. Cela pourrait permettre à des attaquants d'injecter du code malveillant dans les données stockées.
- **Taille limitée** : il y a généralement une limite de taille (généralement quelques mégaoctets) par domaine. Stocker de grandes quantités de données peut affecter les performances du site web.
- **Utilisation sécurisée du HTTPS** : Assurez-vous que votre site web utilise HTTPS pour chiffrer les données échangées entre le navigateur de l'utilisateur et votre serveur. Cela réduit le risque d'interception des données stockées.
- **Gestion des autorisations** : Les navigateurs modernes demandent généralement la permission de l'utilisateur avant de stocker des données. Assurez-vous de respecter les préférences de l'utilisateur et de ne stocker que les données nécessaires.
- **Nettoyage des données** : Évitez d'accumuler des données obsolètes ou inutiles. Assurez-vous de nettoyer régulièrement les données qui ne sont plus nécessaires pour réduire les risques en cas de compromission.

# DANS L'INSPECTEUR

Depuis l'inspecteur de votre navigateur, vous pouvez avoir accès aux différents espaces de stockage.

Depuis la console, vous pouvez également manipuler le `localStorage` ou la `sessionStorage` au travers des méthodes disponibles :

- `getItem()`
- `removeItem()`
- `setItem()`
- `length()`
- `clear()`



# LOCALSTORAGE VS SESSIONSTORAGE

	localStorage	sessionStorage
Type d'objet	Objet JavaScript	
Portée	Domaine du site	Session de navigation
Durée de stockage	Persistant (reste jusqu'à ce qu'il soit explicitement supprimé)	Temporaire (disponible uniquement pendant la durée de la session)
Volume de stockage	Plusieurs Mo (selon le navigateur)	
Accessibilité	Tous les scripts de la même origine (domaine, protocole et port)	
Persistance	Persistant, reste jusqu'à ce qu'il soit explicitement supprimé	Temporaire, disparaît lorsque la session se termine (par exemple, lorsque l'onglet ou le navigateur est fermé)
Utilisation typique	Stockage de données utilisateur, telles que des préférences ou des données de configuration	Stockage de données temporaires nécessaires pendant la session de navigation de l'utilisateur, telles que des informations de session utilisateur ou des états temporaires
Sécurité	Doit être utilisé avec prudence pour éviter les fuites de données sensibles	
Exemple d'utilisation	Stockage des préférences utilisateur	Stockage des informations de session utilisateur

# EXEMPLE

Reprenons notre exemple précédent en remplaçant la méthode basique par l'utilisation du `formData` et de `localStorage`

- Au niveau du formulaire, sur la soumission, je crée un élément `formData` à partir de mon formulaire.
- Ensuite, afin de pouvoir le transmettre, je dois effectuer une conversion en `objet Javascript` et le transformer en `JSON (JSON.stringify)`.
  - *En effet, le `localStorage` ne peut contenir des objets complexes tel que `formData`.*

```
document
.getElementById("userForm")
.addEventListener("submit", function (event) {
  event.preventDefault();

  let isValid = checkForm();

  if (isValid) {
    // Stocker les informations dans le localStorage

    // creation du formData
    let formData = new FormData(event.target);
    // Affiche les données du formulaire dans la console
    for (let [name, value] of formData.entries()) {
      console.log(`${name}: ${value}`);
    }

    // Convertit les données du formulaire en objet
    // le localStorage ne peut stocker que du texte donc le JSON est parfait
    // Cela implique de transformer l'objet FormData en un objet JavaScript standard,
    // puis de le convertir en chaîne JSON avec JSON.stringify().
    const formDataObject = {};
    formData.forEach((value, key) => {
      formDataObject[key] = value;
    });

    // Sauvegarde les données dans localStorage
    localStorage.setItem("userInfo", JSON.stringify(formDataObject));

    // Rediriger vers la page de compte
    window.location.href = "compte.html";
  }
});
```



## EXEMPLE SUITE

Enfin sur la page de compte, je vais pouvoir récupérer les informations depuis le `localStorage` et reparcourir les données.

1. Lecture depuis le `localStorage` en fonction du nom donné et conversion en objet Javascript (`JSON.parse`)
2. Parcours de l'objet Javascript, pour l'afficher dans les bons éléments HTML.

```
// Récupérer les informations de l'utilisateur depuis le localStorage
// Lorsque vous récupérez les données depuis localStorage,
// vous devez les convertir de JSON en objet JavaScript avec JSON.parse().
const userInfoFormData = JSON.parse(localStorage.getItem("userInfo"));

// contrôle
console.info(userInfoFormData);

// si l'objet est bien présent dans le localStorage
// mise à jour des éléments html
if (userInfoFormData) {

    // parcours du formData
    for(const pair of Object.entries(userInfoFormData)) {

        // contrôle
        console.table(pair);

        // ciblage de l'élément correspondant
        const tag = document.getElementById(pair[0]);
        // remplissage de la valeur de l'élément
        tag.textContent = pair[1];
    }

} else {
    alert( 'Aucune donnée de formulaire trouvée.' );
}
```

# EXEMPLE DE CLASSE UTILITAIRE

Voici un exemple de classe Javascript contenant des méthodes d'accès sur [localStorage](#) et [sessionStorage](#).

Pour pouvoir utiliser ces méthodes, il suffit de les importer :

```
// Importer les fonctions du module de stockage
import {
  saveToLocalStorage,
  saveToSessionStorage,
  getAllLocalStorageItems,
  getAllSessionStorageItems,
  clearLocalStorage,
  clearSessionStorage
} from './storage-utils.js';
```

```
// Fonctions utilitaires pour la gestion du stockage
export function saveToLocalStorage(key, data) {
  localStorage.setItem(key, JSON.stringify({
    ...data,
    date: new Date().toLocaleString()
  }));
}

export function saveToSessionStorage(key, data) {
  sessionStorage.setItem(key, JSON.stringify({
    ...data,
    date: new Date().toLocaleString()
  }));
}

export function getAllLocalStorageItems() {
  return Object.keys(localStorage)
    .map(key => ({
      key,
      value: JSON.parse(localStorage.getItem(key))
    }));
}

export function getAllSessionStorageItems() {
  return Object.keys(sessionStorage)
    .map(key => ({
      key,
      value: JSON.parse(sessionStorage.getItem(key))
    }));
}

export function clearLocalStorage() {
  localStorage.clear();
}

export function clearSessionStorage() {
  sessionStorage.clear();
}
```