Projet 9 -A : Base Javascript et API REST Fetch()

Ressources :

https://github1s.com/michelonlineformapro/Projet9-A-API-Javascript-Vanilla

Reprise des cours de Pierre GIRAUD : https://www.pierre-giraud.com/javascript-apprendre-coder-cours/

- **INTRODUCTION AU COURS JAVASCRIPT**

1. Introduction au JavaScript
2. L'environnement de travail pour ce cours JavaScript
3. Où écrire le code JavaScript ?
4. Commentaires, indentation et syntaxe de base en JavaScript

- **LES VARIABLES ET TYPES DE VALEURS JAVASCRIPT**

1. Présentation des variables JavaScript
2. Les types de données en JavaScript
3. Présentation des opérateurs arithmétiques et d'affectation JavaScript
4. La concaténation et les littéraux de gabarits en JavaScript
5. Les constantes en JavaScript

- **LES STRUCTURES DE CONTRÔLE JAVASCRIPT**

1. Structures de contrôle, conditions et opérateurs de comparaison JavaScript
2. Les conditions if, if…else et if…else if…else en JavaScript
3. Opérateurs logiques, précédence et règles d'associativité des opérateurs en JavaScript
4. Utiliser l'opérateur ternaire pour écrire des conditions JavaScript condensées
5. L'instruction switch en JavaScript
6. Présentation des boucles et des opérateurs d'incrémentation et de décrémentation en JavaScript
7. Les boucles while, do… while, for et for… in et les instructions break et continue en JavaScript

- **LES FONCTIONS EN JAVASCRIPT**

1. Présentation des fonctions JavaScript
2. Portée des variables et valeurs de retour des fonctions en JavaScript
3. Fonctions anonymes, auto-invoquées et récursives en JavaScript

- **L'ORIENTÉ OBJET EN JAVASCRIPT**

1. Introduction à l'orienté objet en JavaScript
2. Création d'un objet JavaScript littéral et manipulation de ses membres
3. Définition et création d'un constructeur d'objets en JavaScript
4. Constructeur Object, prototype et héritage en JavaScript
5. Les classes en JavaScript

- **VALEURS PRIMITIVES ET OBJETS GLOBAUX JAVASCRIPT**

1. Valeurs primitives et objets prédéfinis en JavaScript
2. L'objet global JavaScript String, propriétés et méthodes
3. L'objet global JavaScript Number, propriétés et méthodes
4. L'objet global JavaScript Math, propriétés et méthodes
5. Les tableaux en JavaScript et l'objet global Array
6. Les dates en JavaScript et l'objet global Date

- **MANIPULATION DU BOM EN JAVASCRIPT**

1. JavaScript API, Browser Object Model et interface Window

2. L’interface et l’objet Navigator et la géolocalisation en JavaScript
3. L’interface et l’objet History en JavaScript
4. L’interface et l’objet Location en JavaScript
5. L’interface et l’objet Screen en JavaScript

- **MANIPULATION DU DOM EN JAVASCRIPT**

1. Présentation du DOM HTML et de ses APIs accessibles en JavaScript
2. Accéder aux éléments dans un document avec JavaScript et modifier leur contenu
3. Naviguer ou se déplacer dans le DOM en JavaScript grâce aux noeuds
4. Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript
5. Manipuler les attributs et les styles des éléments via le DOM en JavaScript
6. La gestion d’évènements en JavaScript et la méthode addEventListener
7. La propagation des évènements en JavaScript
8. Empêcher un évènement de se propager et annuler son comportement par défaut en JavaScript

- **UTILISATION DES EXPRESSIONS RÉGULIÈRES EN JAVASCRIPT**

1. Introduction aux expressions régulières ou expressions rationnelles en JavaScript
2. Utiliser les expressions régulières pour effectuer des recherches et remplacements en JavaScript
3. Les classes de caractères et classes abrégées des expressions régulières JavaScript
4. Les métacaractères point, alternatives, ancres et quantificateurs des expressions régulières JavaScript
5. Créer des sous masques et des assertions dans les expressions régulières JavaScript
6. Les drapeaux, options ou marqueurs des expressions régulières JavaScript

- **NOTIONS AVANCÉES SUR LES FONCTIONS JAVASCRIPT**

1. Paramètres du reste et opérateur de décomposition des fonctions JavaScript
2. Les fonctions fléchées JavaScript
3. Les closures en JavaScript
4. Gestion du délai d’exécution en JavaScript avec setTimeout() et setInterval()

- **GESTION DES ERREURS ET MODE STRICT EN JAVASCRIPT**

1. Gestion des erreurs en JavaScript
2. Le mode strict en JavaScript

- **L’ASYNCHRONE EN JAVASCRIPT**

1. Introduction à l’asynchrone en JavaScript
2. Les promesses en JavaScript
3. Utiliser async et await pour créer des promesses plus lisibles en JavaScript
4. Le chemin critique du rendu et les attributs HTML async et defer

- **SYMBOLES, ITÉRATEURS ET GÉNÉRATEURS EN JAVASCRIPT**

1. Les symboles et l’objet Symbol en JavaScript
2. Les protocoles et objets Itérable et Itérateur en JavaScript
3. Les générateurs en Javascript

- **STOCKAGE DE DONNÉES DANS LE NAVIGATEUR EN JAVASCRIPT**

1. Les cookies en JavaScript
2. L’API Web Storage : local Storage et session Storage en JavaScript
3. Utiliser l’API de stockage IndexedDB en JavaScript

- **L’ÉLÉMENT HTML CANVAS ET L’API CANVAS**

1. Présentation de l’élément HTML Canvas et de l’API Canvas
2. Dessiner des rectangles dans un élément HTML Canvas en Javascript
3. Définir des tracés pour dessiner des formes dans un canevas en JavaScript

4. Création de dégradés ou de motifs dans un canevas en JavaScript
5. Ajout d'ombres et utilisation de la transparence dans un canevas en JavaScript
6. Ajouter du texte ou une image dans un canevas en JavaScript
7. Appliquer des transformations sur un canevas en JavaScript

- **LES MODULES JAVASCRIPT**

1. Les modules JavaScript : import et export

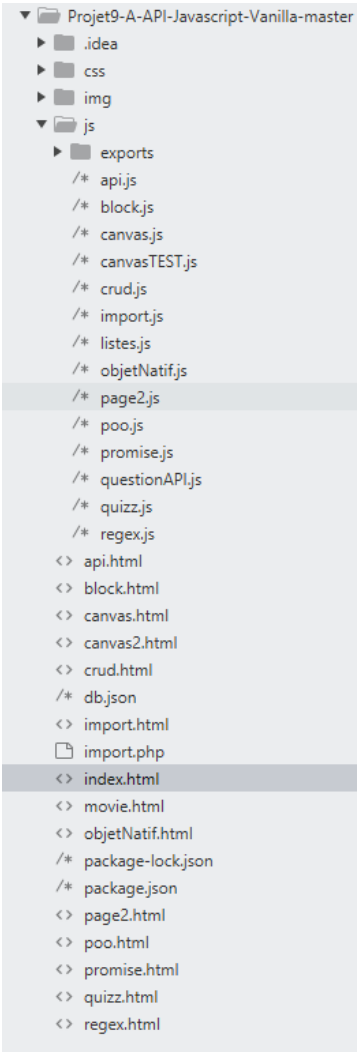
- **JSON, AJAX ET FETCH EN JAVASCRIPT**

1. Présentation de JSON et utilisation en JavaScript
2. Introduction à l'Ajax en JavaScript
3. Créer des requêtes Ajax en utilisant l'objet XMLHttpRequest en JavaScript
4. Présentation et utilisation de l'API Fetch en Javascript

- **CONCLUSION DU COURS COMPLET JAVASCRIPT**

1. Conclusion du cours complet

Objectif : Suivre le cours de Pierre GIRAUD a traves de multiple exemple à retrouver sur :



1. Une todo liste (liste des taches Javascript)

listes.js

```
//Recup du btn pour ajouter une tache (recup par attrubut id="")

let btnAddTask = document.getElementById('btn-add-task');

//Le bouton declenche un evenement au click

btnAddTask.addEventListener('click', (event) => {
    //On supprime le coportement pad defaut
    event.preventDefault();

    //Input text des taches on recupère l'elemnt par id et sa valeur <input id="" value=""/>
    let inputTask = document.querySelector("#input-task").value;
    //Element parent <tbody> de html <table>
    //La méthode querySelector() de l'interface Document retourne le premier Element dans le document
    // correspondant au sélecteur - ou groupe de sélecteurs - spécifié(s), ou null si aucune correspondance n'est trouvée.
    let tBody = document.querySelector('tbody');
    //Creer un tr (table row)
    let tr = document.createElement('tr');
    //Ajout d'une classe au <tr>
    tr.className = 'mt-3';
    //creer un table data <td>
    let td = document.createElement('td');

    //Ajouter un btn supprimer la taches
    let btnDelete = document.createElement('button');
    //Ajout d'une classe lib materialize
    btnDelete.className = 'right-align';
    //Ajout d'une croix dans le bouton avec createTextNode
    let btnDeleteText = document.createTextNode('\u00D7');
    //Ajout d'une classe materialize
    btnDelete.className = "right-align btn-floating btn-large waves-effect waves-light red";
    //Ajout de la croix dans le bouton
    btnDelete.appendChild(btnDeleteText);

    //Verif que le champs n'est pas vide
    if(inputTask === ''){
        M.toast({html: 'Merci de remplir le champs taches'})
    }else{
        //Traitement ajout de la taches
        tBody.appendChild(tr)
        //Enfant de <tr>
        tr.appendChild(td)
        //Debug pour test
        console.log(inputTask);
        //Ajout du bouton supprimer
        td.innerHTML = inputTask;
        //A chaque ajour de la tache on ajoute un bouton
        td.append(btnDelete);
        //Au clcik sur le bouton supprimer on declenche du css (la ligne disparait)
        btnDelete.addEventListener('click' , () => {
            td.style.display = 'none';
        })
    }

    //On vide le champs input a chaque ajout
    document.getElementById('input-task').value = '';
})
```

2. Créer, exporter et importer des modules avec JavaScript

CLASSE & EXPORT :

```
export function Personnages(nom, email, age, sexe){
    this.nom = nom;
    this.email = email;
    this.age = age;
    this.sexe = sexe;

    this.creerPersonnage = () => {
        let personnageContainer = document.createElement('div');

        personnageContainer.innerHTML =
            `
            <ul class="collection">
                <li class="collection-item">Nom: ${this.nom}</li>
                <li class="collection-item">Email: ${this.email}</li>
                <li class="collection-item">Age: ${this.age}</li>
                <li class="collection-item">Sexe: ${this.sexe}</li>
            </ul>
            `
        document.body.append(personnageContainer)
    }
}
```

IMPORT DE MODULES :

```
import {Personnages} from "./exports/personnes.class";

let perso1 = new Personnages('Michael', 'mic@hotmail.fr', 65, 'Homme');
let perso2 = new Personnages('Sophie', 'sophie@cool.fr', 45, 'Femme');


perso1.creerPersonnage();
perso2.creerPersonnage();
```

3. La programmation Orientée Objet avec JavaScript, exemple d'objet littéral, des fonctions et des classes
    a. Objet Littéral :

```
//Un objet literal = un objet avec des valeurs a l'interieur de ce dernier
let literalPersonne = {
    //Clé / Valeur
    nom: ['MICHEL', 'Michael'],
    age: 35,
    email: 'micpiwo@hotmail.fr',

    //Fonction dans un objet
    direBonjour(){
        let literalContainer = document.getElementById('literalPersonne')
        literalContainer.innerHTML = `
            Bonjour : ${this.nom[0]} ${this.nom[1]}
            <p>Tu as : ${this.age} ans</p>
            <p>Et ton email est : ${this.email}</p>
        `
    }
}

//Appel de notre objet et de sa fonction :
literalPersonne.direBonjour();

//on peu modifier les propriétées de l'objet = ici l'age
let changeAge = literalPersonne.age = 40;
//Test de debug
console.log(changeAge);

//On peu creer une nouvelle fonction avec cet objet

let newFunction  = () => {
    let chanegContainer = document.getElementById('changePersonne')
    chanegContainer.innerHTML = `
            Bonjour : ${literalPersonne.nom[0]} ${literalPersonne.nom[1]}
            <p>Tu as : ${literalPersonne.age} ans</p>
            <p>Et ton email est : ${literalPersonne.email}</p>
            <p>OU BIEN ${literalPersonne['email']}</p>
        `
}

//Appel de la nouvelle fonction
newFunction();
```

b. Objet dynamique

```
//Creer des objets a la chaine et dynamique -> ceci evite de creer 2 objet literaux
let objetDynamique = document.getElementById('objetDynamique');

//La fonction prend des paramètres c une fonction constructeur
function Users(nom, age, email){
    this.nom = nom;
    this.age = age;
    this.email = email;

    //Creation d'une fonction dans une fonction

    this.saluer = function(){
        objetDynamique.innerHTML +=
            `
                <p>Un objet dynamique = ${this.nom[0]} et le prenom ${this.nom[1]}</p>
                <p>Tu as : ${this.age} ans</p>
                <p>Et ton email est : ${this.email}</p>
            `
    }

}


//Creation de variable et instance de objet + valeur
let utilisateur1 = new Users(['michael', 'michel'], 45, 'test@test.fr');

let utilisateur2 = new Users(['michael2', 'michel2'], 41, 'test@test.fr');

//Afficher les objets = appel de la variable + la fonction de la classe
utilisateur1.saluer();
//IMPOSIIBLE DE RAPELLER LA MEME METHODE
utilisateur2.saluer();
```

c. Classes

poo.js

```
//Utiliser une classe et reutiliser une methode

class Personnage{
    //Creation d'un constructor
    constructor(nom, age, email) {
        this.nom = nom;
        this.age = age;
        this.email = email
    }
    //Creation de la fonction pour afficher des personnages
    getPersonnage(){
        let objetClasse = document.getElementById('objetClasse')
        objetClasse.innerHTML +=
            `
                <p>Une classe nom = ${this.nom[0]} et le prenom ${this.nom[1]}</p>
                <p>Tu as : ${this.age} ans</p>
                <p>Et ton email est : ${this.email}</p>
            `
    }
}



//Creation de variable = instance de notre classe
let perso1 = new Personnage(['Laurent', 'TOUVABIEN'], 78, 'laurent@tes.com');
let perso2 = new Personnage(['Bob', 'LAGADECK'], 78, 'bob@tes.com');
let perso3 = new Personnage(['Annie', 'FAIDUVELO'], 78, 'annie@tes.com');

//A partir de la classe intancié on appel la methode de la classe
perso1.getPersonnage();
perso2.getPersonnage();
perso3.getPersonnage()

//Heritage = ici la classe chien herite de la classe Personnage
class Chien extends Personnage{
    //Creation d'un constructor avec repiser des paramètres du constructeur parent + ajout d'un paramètre
    constructor(nom, age, email, race) {
        //Recuperation du constructor parent avec super()
        super(nom, age, email);
        this.race = race;
    }
    //Creation d'une methode pour afficher le chien au personnage
    setRaceChien(){
        let heritageChien = document.getElementById('heritageChien')
        heritageChien.innerHTML +=
            `
                <p>Une classe nom = ${this.nom[0]} et le prenom ${this.nom[1]}</p>
                <p>Tu as : ${this.age} ans</p>
                <p>Et ton email est : ${this.email}</p>
                <p>Race du chien =  ${this.race}</p>
            `

    }
}

//Variable + insatnce de la classe chien qui herite de Personnage

let perso4 = new Chien('Jojo', 78, 'chien@chien.com', 'Caniche');
//Appel de la methode de la classe chien
perso4.setRaceChien()
```

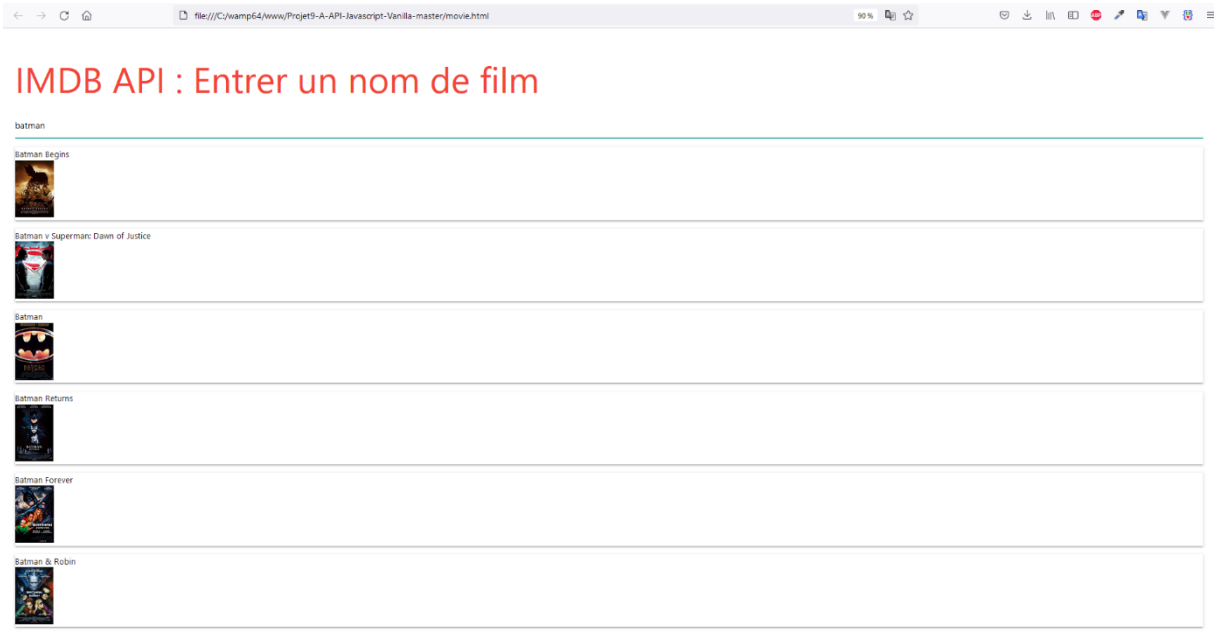
4. Introduction au RegEx et des formulaires avec JavaScript

```
function motDePasse(){

    //Le score
    let score = 0;
    //La saisie
    let motPasse = document.querySelector('#password').value
    //Les elements
    let minuscule = document.querySelector('#minuscule');
    let majuscule = document.querySelector('#majuscule');
    let chiffre = document.querySelector('#chiffre');
    let special = document.querySelector('#special');
    let longueur = document.querySelector('#longeur')
    console.log(motPasse)
    //check miniscule
    //Les regex true ou false soit string soit new Regex()
    if(/[a-z]/.test(motPasse)){
        console.log('minuscule = good');
        score++
        minuscule.classList.replace('rouge', 'vert');
    }else{
        console.log('minuscule = no good')
        minuscule.classList.replace('vert', 'rouge');
    }


    //Majsucule

    if(/[A-Z]/.test(motPasse)){
        console.log('majuscule = good');
        score++
        majuscule.classList.replace('rouge', 'vert');
    }else{
        console.log('majuscule = no good')
        majuscule.classList.replace('vert', 'rouge');
    }

    //Chiffre
    if(/[0-9]/.test(motPasse)){
        score++
        chiffre.classList.replace('rouge', 'vert');
    }else{

        chiffre.classList.replace('vert', 'rouge');
    }
    //Caractéres speciaux = ici pas \ car ca echape les caratères
    if(/[$!@/&%#ç]/.test(motPasse)){
        console.log('special = good');
        score++
        special.classList.replace('rouge', 'vert');
    }else{
        console.log('special = no good')
        special.classList.replace('vert', 'rouge');
    }
    //le 6 lettres
    if(motPasse.length >= 6){
        score++
        longueur.classList.replace('rouge', 'vert');
    }else{

        longueur.classList.replace('vert', 'rouge');
    }

    let validBtn = document.getElementById('btn-valid-form')

    if(score === 5){
        validBtn.style.display = 'block';
    }else{
        validBtn.style.display = 'none';
    }


}
```

5. Créer des éléments avec la balise <canvas>

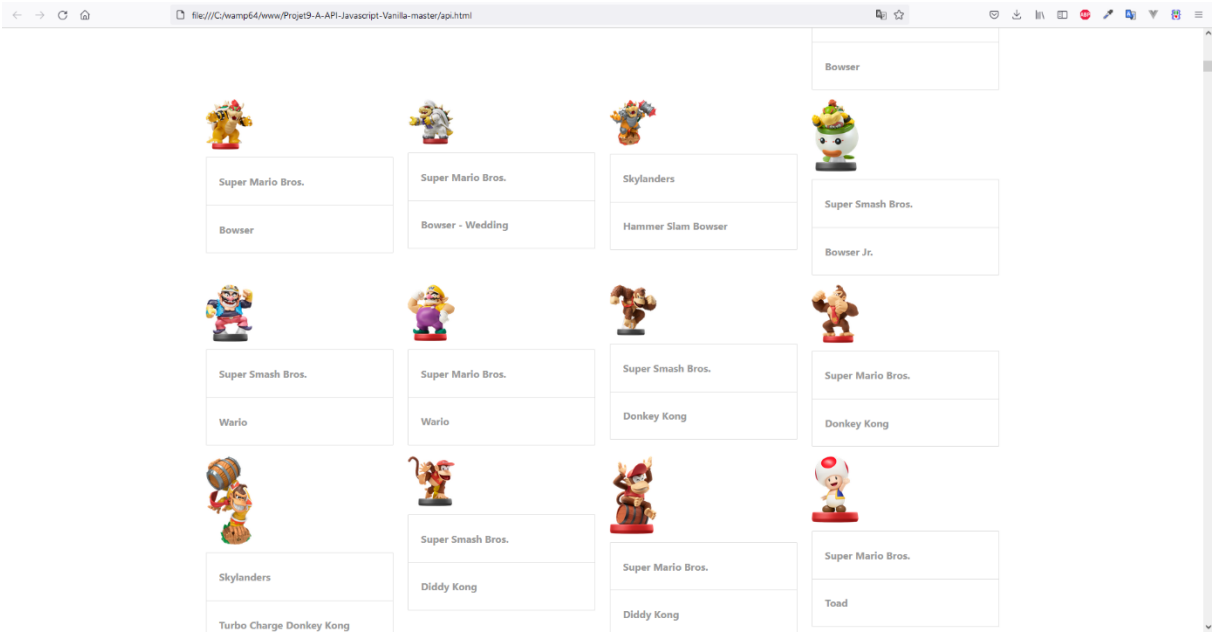
- //SOURCE = https://jsfiddle.net/rtoal/wvp4scLL/
- Js/canvas.js

6. Appeler et consommer une API avec AJAX et jQuery : (recherche de film avec omdbapi)



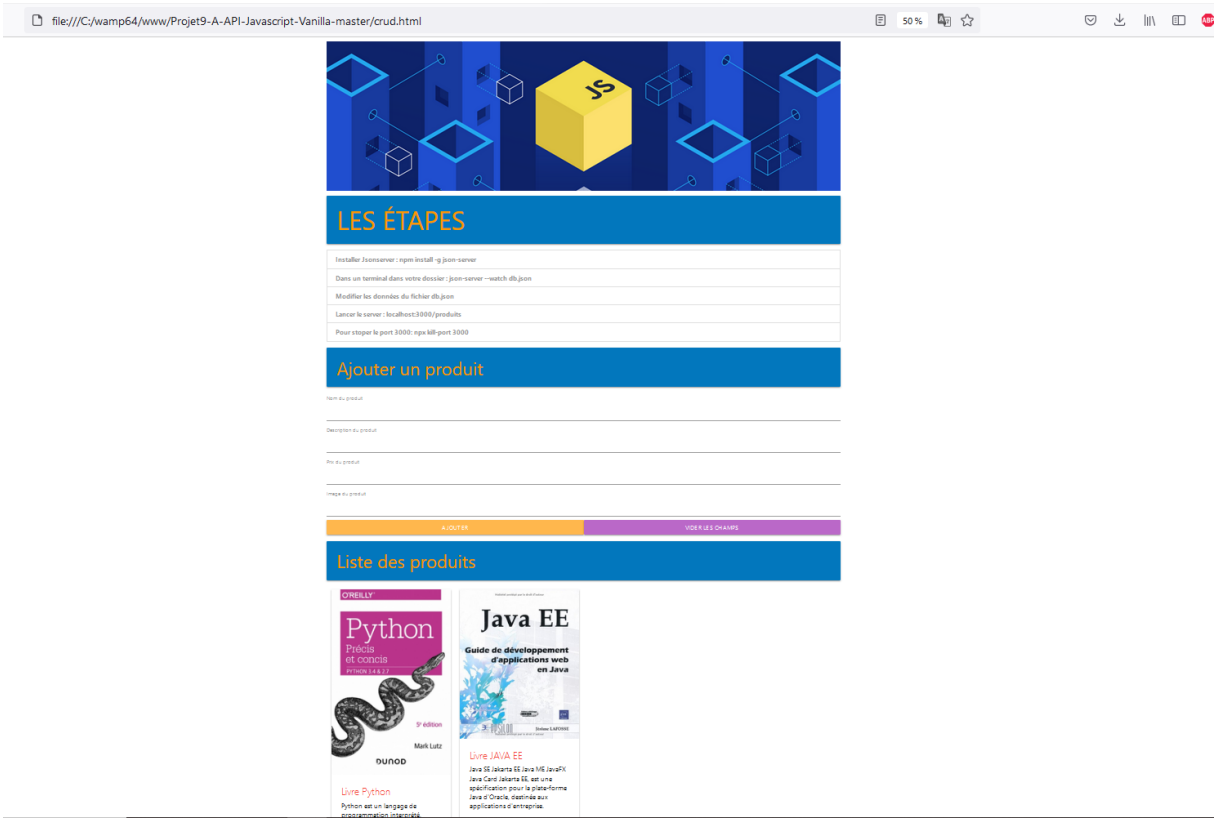

```
<script src="https://code.jquery.com/jquery-3.3.1.min.js" crossorigin="anonymous"></script>
<script>
  $(function() {

    const search = $('#search');
    const omdbApiKey = 'd213cd39';

    search.on('input', function() {
      console.log( $(this).val() );

      $.ajax({
        url: 'http://www.omdbapi.com/',
        type : 'GET',
        data : {
          'apikey': omdbApiKey,
          's' : $(this).val()
        },

        success : function(result) {

          if (search.val().length >= 3 ) {

            $('#results').html('');

            if (result.Search && result.Search.length > 0) {


              // comme en PHP : foreach ($result['Search'] as $movie)
              result.Search.forEach(function (movie) {

                console.log(movie);

                let movieHtml = '<div class="card">' +
                        '<div class="card-header">' + movie.Title + '</div>' +
                        '<div class="card-body"><img src="' + movie.Poster + '" height="100"></div>' +
                        '</div>';

                $('#results').append(movieHtml);
              });
            }
          }
        },

        // error : function(result) {
        //     console.log('erreur réseau !');
        // },

        // complete: function(result) {
        //     console.log('fini !');
        // }
      });
    });
  });
</script>
```

7. Introduction a fetch() javascript, création de promesse et test via Postman (afficher des amiiboo via l'API Nintendo)



```
o.js                    api.js
document.addEventListener('DOMContentLoaded', () => {
    const amiibo = document.querySelector('#amiiboContainer');

    fetch('https://www.amiiboapi.com/api/amiibo/',{
        method: 'GET',
        headers:{
            'Access-Control-Allow-Origin': '*',
            'Content-Type': 'application/json'
        },
    })
        .then(reponse => reponse.json())
        .then(result => result.amiibo.forEach(afficherAmiibo))
        .catch(error => console.log('erreur', error))


    function afficherAmiibo(amiiboData){
        //creation d'un div
        const amiiboDIV = document.createElement('div');
        amiiboDIV.className = 'col s3 m3';
        amiiboDIV.innerHTML = `
            <img src="${amiiboData.image}" alt="" title="" width="25%" />
            <ul class="collection">
                <li class="collection-item">
                    <p>${amiiboData.amiiboSeries}</p>
                </li>
                <li class="collection-item">
                    <p>${amiiboData.name}</p>
                </li>
            </ul>
        `
        amiibo.appendChild(amiiboDIV);
    }

})
```

8. Crud Javascript avec json-server et test des 4 méthodes (GET, POST, PUT-PATCH et DELETE) API REST (crud.js)




```
//Charger le DOM
document.addEventListener('DOMContentLoaded', () => {
    //Conteneur de produit id HTML
    const produits = document.querySelector('#produits');
    //Recuperer id du formulaire + submit sur le bouton
    const produitForm = document.querySelector('#ajouter-produit-form');
    //Declenche un evenement au click sur le bouton valider du formulaire ajouter
    produitForm.addEventListener('submit', ajouterProduit);

    //Conteneur du formulaire d'edition
    const updateForm = document.querySelector('#updateForm');

    /******************************** METHODE GET JSON LOCALHOST:3000 **************************/

    //Methode fetch + url + methode GET + options
    fetch('http://localhost:3000/produits', {
        method: "GET",
        headers: {
            //Autorisé CORS
            'Access-Control-Allow-Origin': '*',
            //Type de contenu
            'Content-Type': 'application/json'
        }
    })
        //Promise + recup des données au format Json
        .then(response => response.json())
        //Boucle de lecture (le conteneur + nom de la collection) + appel de la fonction ajouterProduit
        .then(produits => produits.forEach(afficherProduit))

    /*********************************AFFICHER LES CARTES DE PRODUIT**********************/
```