

Devoir

abdcourses



Table des matières :

Sujet :	2
Outils utiliser :	2

Introduction	3
Javascript :	3
Le mécanisme :	3
Les expressions régulières :	4
AddEventListener :	5
Le DOM :	6
Map () ;	7
Filter() :	7
Arrow function :	8
Bootstrap :	9
Explication du code :	10
Page sign up :	10
Partie html et style :	10
Partie interactivité :	12
Username :	12
Password :	13
Password confirmation :	15
Verification des champs :	15
Navbar :	19
Page index :	20
Partie html :	20
Partie interactivité :	22
List des cours :	22
Générer des cours aléatoirement :	23
Affichage de cours :	24
Courses page :	25
Filtrage :	25
Affichage :	30
Page about :	30
Conclusion :	31
Bibliographie :	31

Sujet :

Notre projet consiste à créer un site web de vente de cours de développement web. Il s'adressera aux développeurs débutants et intermédiaires souhaitant acquérir de nouvelles compétences et se perfectionner dans le développement web.

Le site proposera une variété de cours sur les technologies et les langages de développement web les plus populaires, tels que HTML, CSS, JavaScript, PHP, etc.

Les utilisateurs pourront s'inscrire, sélectionner les leçons qui les intéressent. Ils peuvent aussi utiliser notre système de filtration des leçons par prix, catégorie et langage de programmation pour trouver celles qui correspondent le mieux à vos besoins et à votre budget.

Notre objectif est de créer un site web convivial, intuitif et performant, qui réponde aux besoins et aux attentes des utilisateurs en matière de "développement web".

Outils utiliser :

Introduction

Notre application a été développée en utilisant plusieurs outils et technologies de pointe. Nous avons utilisé JavaScript, HTML, CSS et Bootstrap, et nous allons couvrir chaque une de ces technologies de manière théorique et pratique. Nous allons expliquer les concepts fondamentaux de chaque technologie, ainsi que montrer comment les utiliser pour voir comment on réussit de réaliser cette plateforme.

Javascript :

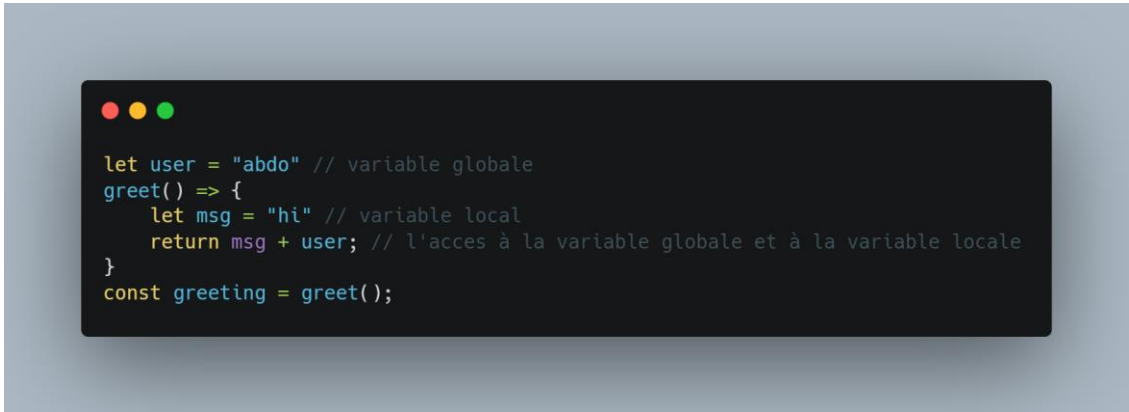
JavaScript est un langage de programmation côté client utilisé pour créer des sites web interactifs et des applications web. Il est exécuté côté client et permet de réagir rapidement aux actions de l'utilisateur en offrant des interactions et des effets sur les pages web, comme des animations et des formulaires interactifs. JavaScript est orienté objet et utilise des concepts tels que les variables, les fonctions et les objets pour structurer et exécuter du code. Il est souvent utilisé en conjonction avec HTML et CSS pour créer des sites web attrayants et dynamiques.

Le mécanisme :

En JavaScript, le contexte d'exécution désigne l'environnement dans lequel du code est exécuté. Il s'agit d'un concept important à comprendre lorsque vous travaillez avec ce langage, car il détermine comment vous pouvez accéder aux variables et aux fonctions depuis différentes parties de votre code.

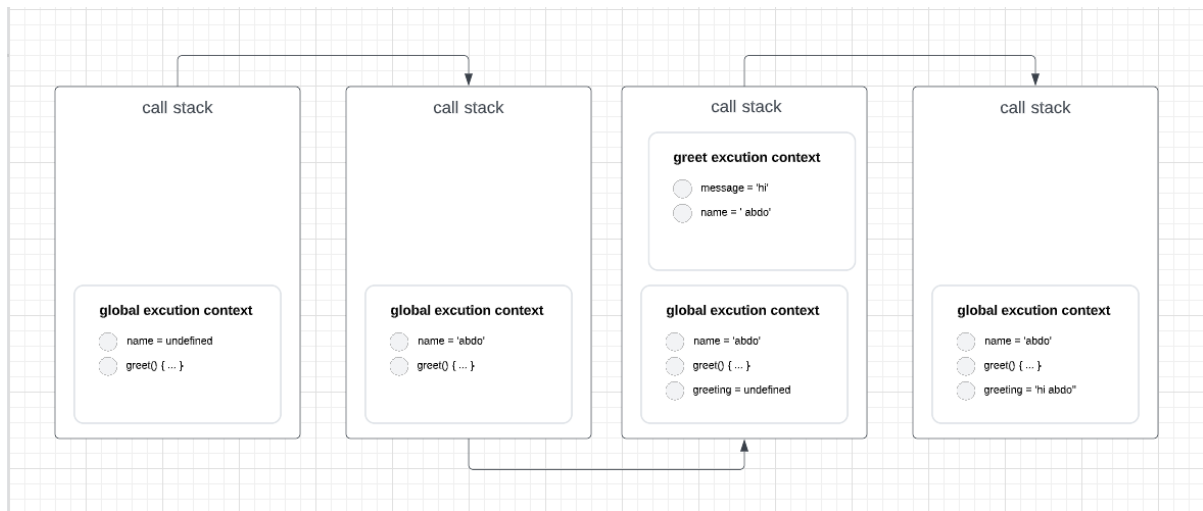
Il existe deux types de contextes d'exécution en JavaScript : le contexte global et le contexte local. Le contexte global est l'environnement d'exécution qui est disponible partout dans votre code et inclut les variables et fonctions définies à la racine de votre script ainsi que les objets globaux tels que **window** . Le contexte local, quant à lui, est créé chaque fois qu'une fonction est appelée et inclut les variables et fonctions définies à l'intérieur de cette fonction, ainsi que tous les paramètres passés à la fonction.

Voici un exemple de code qui illustre le fonctionnement des contextes d'exécution en JavaScript :



```
let user = "abdo" // variable globale
greet() => {
  let msg = "hi" // variable locale
  return msg + user; // l'accès à la variable globale et à la variable locale
}
const greeting = greet();
```

Dans cet exemple, la variable **user** est définie dans le contexte global et est donc disponible partout dans le code. La fonction **greet** définit une variable **msg** dans son contexte local et utilise les deux variables dans son code. Lorsque la fonction **greet** est appelée, le code est exécuté dans le contexte local de la fonction et a accès à la variable **msg**, mais aussi à la variable **user** du contexte global.



Maintenant que nous avons compris comment fonctionne JavaScript, nous allons passer directement aux fonctions JavaScript nécessaires pour construire ce site web.

Les expressions régulières :

Les expressions régulières (ou regex pour short) sont une syntaxe spéciale utilisée pour vérifier si une chaîne de caractères correspond à un certain modèle ou pour remplacer certaines parties d'une chaîne de caractères par d'autres.

En JavaScript, vous pouvez créer une expression régulière en utilisant les slashes / pour entourer l'expression et en utilisant des caractères spéciaux pour définir le modèle de la chaîne de caractères. Par exemple, l'expression régulière `/abdcourses/` correspond à toute chaîne de caractères qui contient les caractères **abdcourses** dans cet ordre.

Voici quelques exemples d'utilisation des expressions régulières en JavaScript :

- Vérifier si une chaîne de caractères correspond à un modèle :



```
let regex = /abdcourses/;  
let str = 'abdcourses';  
  
console.log(regex.test(str)); // true
```

Il existe de nombreuses autres opérations que vous pouvez effectuer avec les expressions régulières et par exemple on effectue le contrôle de validation de notre page d'inscription "sign up". Si vous souhaitez en savoir plus sur ce sujet, je vous recommande de consulter la documentation de [JavaScript](#) Sur [les expressions régulières](#).

AddEventListener :

addEventListener est une méthode JavaScript qui permet de définir une fonction à exécuter lorsqu'un événement spécifique se produit sur un élément HTML. Elle prend en paramètres le nom de l'événement (par exemple **click**, **submit**, **focus**, etc.), ainsi que la fonction à exécuter lorsque l'événement se produit.

Voici un exemple d'utilisation de **addEventListener** qui affiche un message lorsque l'utilisateur clique sur un bouton :



```
const button = document.querySelector('button');  
  
button.addEventListener('click', () => {  
  console.log('Button clicked!');  
});
```

Le DOM :

Le Document Object Model (DOM) est un modèle de représentation de documents HTML et XML en mémoire. Il permet aux développeurs de manipuler les éléments d'un document numérique comme s'ils étaient des objets dans un programme informatique.

Le DOM est organisé en arborescence, avec chaque élément du document étant représenté par un objet. Les objets peuvent avoir des propriétés (comme l'attribut "class" d'une balise HTML) et des méthodes (comme "getElementById" qui permet de récupérer un élément du document à partir de son identifiant unique).


Le DOM est utilisé en conjonction avec le langage de programmation JavaScript pour permettre une interaction en temps réel avec le contenu d'un document HTML ou XML. Par exemple, un développeur peut utiliser le DOM pour ajouter ou supprimer des éléments d'un document, ou pour changer le style ou le contenu de ces éléments.

Fonctions utiles pour manipuler les tableaux :

Map () ;

La méthode **map()** de JavaScript est une fonction qui permet de transformer chaque élément d'un tableau en appliquant une fonction de transformation. Elle retourne un nouveau tableau contenant les résultats de l'application de la fonction de transformation à chaque élément du tableau original.

Voici comment utiliser la méthode **map()** :




```
const ar = [1, 2, 3, 4];  
const newArr = arr.map(e => e * 2);  
  
console.log(newArr); // [2, 4, 6, 8]
```


Filter() :

La méthode **filter()** de JavaScript est une fonction qui permet de sélectionner un sous-ensemble d'éléments d'un tableau en fonction de certains critères. Elle retourne un nouveau tableau contenant uniquement les éléments qui satisfont les critères de filtrage.

Voici comment utiliser la méthode **filter()** :



```
const arr = [1, 2, 3, 4, 5];  
const newArr = arr.filter(e => e % 2 == 0);  
  
console.log(newArr); // [2, 4]
```

Arrow function :

Les fonctions fléchées (ou "arrow functions" en anglais) sont une syntaxe concise pour écrire des fonctions en JavaScript. Elles ont été introduites avec la spécification ECMAScript 6 (ES6) et sont de plus en plus utilisées dans le développement de code JavaScript moderne.

Les fonctions fléchées se définissent de la manière suivante :



```
const functionName = (parameters) => {  
  // code de la fonction  
}
```

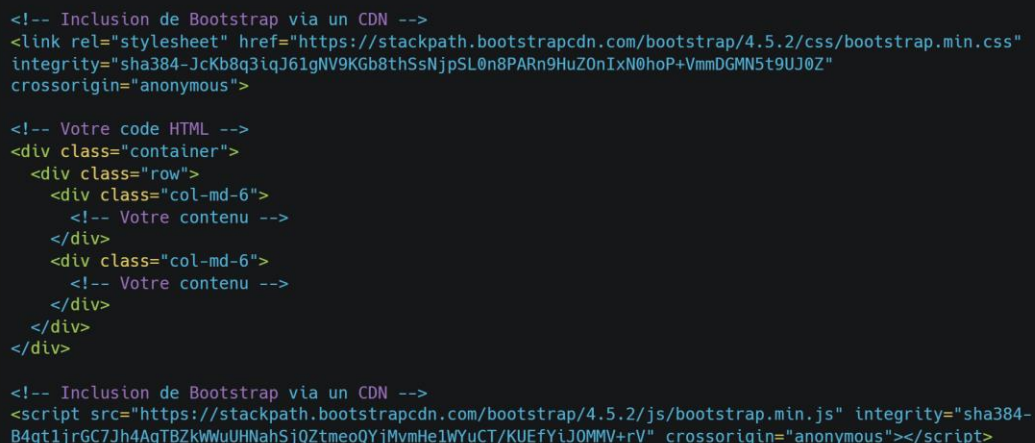
Bootstrap :

Bootstrap est un framework HTML, CSS et JavaScript open-source qui vous permet de développer des sites web et des applications web rapidement et facilement. Il fournit une série de modèles prédéfinis pour la mise en page, la typographie, les formulaires, les boutons, les tableaux, les navbars, les modals, les carousels et de nombreux autres éléments de design.

Bootstrap est basé sur une grille de 12 colonnes qui vous permet de structurer votre contenu de manière flexible et responsive. Il utilise également un système de classes de nommage précis qui vous permet de personnaliser facilement l'apparence de votre site ou de votre application en utilisant des classes prédéfinies.

Pour utiliser Bootstrap dans votre projet, vous pouvez inclure les fichiers CSS et JavaScript de Bootstrap dans votre page web en utilisant les balises `<link>` et `<script>` respectivement. Vous pouvez également utiliser un CDN (Content Delivery Network) pour inclure Bootstrap directement depuis un serveur externe.

Voici un exemple de code HTML qui inclut Bootstrap dans une page web :



```
<!-- Inclusion de Bootstrap via un CDN -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
integrity="sha384-JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"
crossorigin="anonymous">

<!-- Votre code HTML -->
<div class="container">
  <div class="row">
    <div class="col-md-6">
      <!-- Votre contenu -->
    </div>
    <div class="col-md-6">
      <!-- Votre contenu -->
    </div>
  </div>
</div>

<!-- Inclusion de Bootstrap via un CDN -->
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" integrity="sha384-
B4gt1jrGC7Jh4AgTBZkwUuUHNahSjQZtmeoQYjMvmHe1wYUCT/KUEfYiJOMMV+rV" crossorigin="anonymous"></script>
```

En utilisant Bootstrap, vous pouvez facilement créer des sites web et des applications web qui sont esthétiquement agréables, faciles à utiliser et adaptatives à différentes tailles d'écran. Si vous souhaitez en savoir plus sur

Bootstrap, je vous recommande de consulter la documentation en ligne sur le site officiel de [Bootstrap](#).

Explication du code :

Page sign up :

Sign up

Username

Email

Password

Confirm Password

☐ Accept [Condition](#) and [Privacy Policy](#).

Submit



Nous avons réalisé cette page en utilisant Bootstrap pour la partie style et JavaScript pour la partie interactivité.

Puis utilisé des expressions régulières pour s'assurer que l'utilisateur remplissait le formulaire selon les modèles proposés par la plateforme, ainsi que pour s'assurer que les données saisies ne sont pas des requêtes ou des scripts qui pourraient attaquer notre site web ([l'injection SQL ...](#)).

Partie html et style :

```
<div class="container d-flex align-items-start justify-content-center ">
  <div class="w-50 pe-3 pt-5">
    <h1 class="mt-3 pb-3 ">Sign up</h1>
    <div class="form-group pb-2">
      <input type="text" placeholder="Username" class="form-control p-3 border-primary border-2"
id="username">
    </div>
    <div class="form-group pb-2">
      <input type="email" placeholder="Email" class="form-control p-3 border-primary border-2"
id="email">
    </div>
    <div class="form-group pb-2">
      <input type="password" placeholder="Password" class="form-control p-3 border-primary border-2"
id="pwd">
    </div>
    <div class="form-group pb-2">
      <input type="password" placeholder="Confirm Password" class="form-control p-3 border-primary
border-2"
id="ConfPwd">
    </div>
    <input type="checkbox" value="false" class="form-check-input " id="accept" required>
    <label for="accept" class="form-check-label pb-3">Accept <a href="#">Condition</a> and <a
href="#">Privacy
Policy</a> </label>
    <button id="submit" class="btn rounded-pill btn-primary w-50 p-3">Submit</button>
  </div>
  <div class="w-50 h-100">
    
  </div>
</div>
```

Ce code HTML représente une page de formulaire "signup" . La page comprend un formulaire avec plusieurs champs de saisie et un bouton de soumission.

Le formulaire est contenu dans un élément avec la classe **container** et des classes de disposition (**d-flex align-items-start justify-content-center**) qui permettent de disposer les éléments du formulaire et de l'image de manière qu'ils s'affichent côte à côte sur la page.

Le formulaire est divisé en deux parties : un élément contenant les champs de saisie et les boutons et un autre élément contenant une image. Chacun de ces éléments à la classe **w-50**, ce qui signifie qu'ils occuperont chacun 50% de la largeur de la page.

Le formulaire comprend plusieurs champs de saisie, notamment un nom d'utilisateur, une adresse email, un mot de passe et une confirmation du mot de passe. Chacun de ces champs est défini comme un élément **input** de type **text**, **email** ou **password**

Il comprend également un bouton de soumission, qui est défini comme un élément **button** avec l'ID **submit** (pour qu'on peut l'accéder à l'utilisation du **DOM**) et la classe **btn rounded-pill btn-primary w-50 p-3**. Cette classe définit le style du bouton, notamment sa couleur et sa forme arrondie.

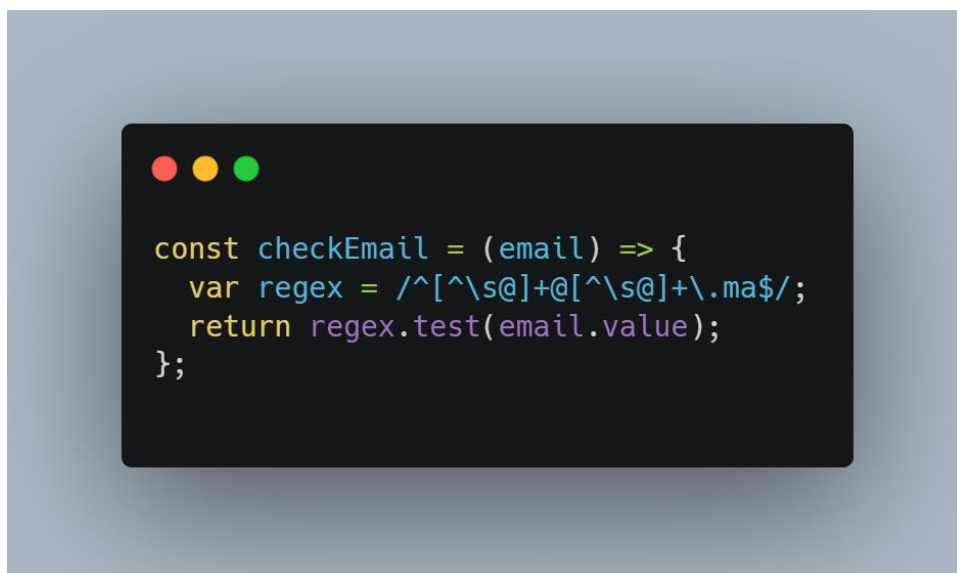
Il y a également un champ de case à cocher qui demande à l'utilisateur d'accepter les conditions et la politique de confidentialité du site Web. Ce champ est défini comme un élément **input** de type **checkbox** avec l'ID **accept** et la classe **form-check-input**. Il est accompagné d'une étiquette (**label**) qui explique ce que l'utilisateur doit faire pour accepter les conditions et la politique de confidentialité.

Enfin, la page comprend une image qui se trouve dans un élément **div** avec la classe **w-50 h-100**. Cette image est définie comme un élément **img** avec la classe **img-fluid w-100**, ce qui signifie qu'elle s'ajustera à la largeur de la page et occupera toute la hauteur disponible.

Partie interactivité :

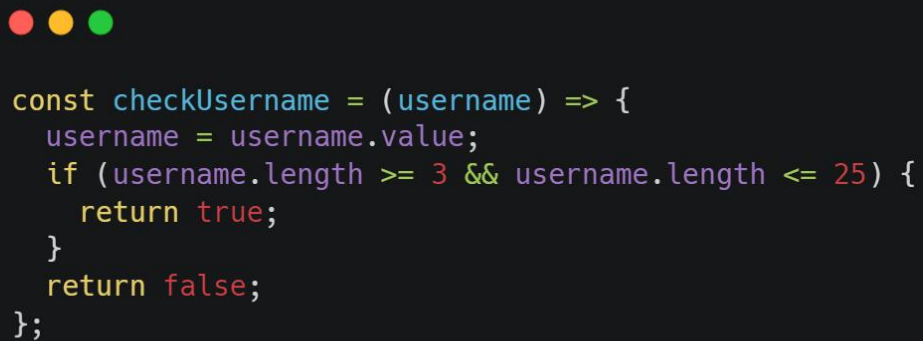
Pour vérifier les champs du formulaire, nous créons des fonctions qui utilisent des expressions régulières.

Email :



La fonction **checkEmail** vérifie si la chaîne passée en paramètre respecte le modèle d'une adresse email qui a la forme **example@example.ma** (une chaîne de caractères qui ne contient pas d'espace, suivie d'un **@**, suivie d'une autre chaîne de caractères qui ne contient pas d'espace, et qui doit se terminer par **.ma**). Elle retourne **true** si la chaîne respecte ce modèle, et **false** sinon.

Username :



```
const checkUsername = (username) => {  
  username = username.value;  
  if (username.length >= 3 && username.length <= 25) {  
    return true;  
  }  
  return false;  
};
```

La fonction **checkUsername** vérifie si la chaîne passée en paramètre a une longueur comprise entre 3 et 25 caractères (inclus). Elle retourne **true** si la chaîne respecte cette condition, et **false** sinon.

Password :

```
const checkPassword = (password) => {  
  if (password.value.length < 8) {  
    return false;  
  }  
  
  var hasUppercase = /[A-Z]/.test(password.value);  
  if (!hasUppercase) {  
    return false;  
  }  
  
  var hasLowercase = /[a-z]/.test(password.value);  
  if (!hasLowercase) {  
    return false;  
  }  
  
  var hasDigit = /\d/.test(password.value);  
  if (!hasDigit) {  
    return false;  
  }  
  
  var hasSpecial = /[!@#$%^&*]/.test(password.value);  
  if (!hasSpecial) {  
    return false;  
  }  
  
  return true;  
};
```

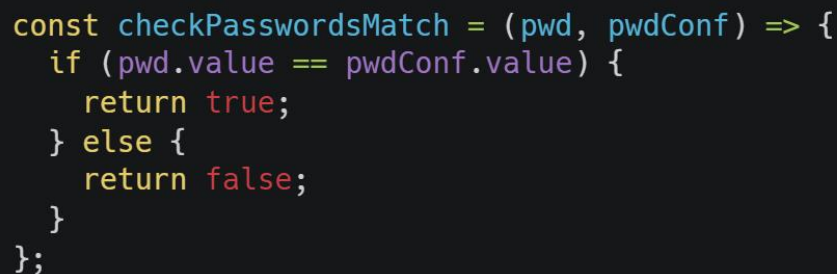
Ce code définit une fonction appelée **checkPassword** qui prend un mot de passe en argument. La fonction vérifie si le mot de passe répond à certains critères et renvoie une valeur booléenne indiquant si le mot de passe est valide.

La fonction vérifie tout d'abord si le mot de passe a au moins 8 caractères. Si le mot de passe est plus court que 8 caractères, la fonction renvoie false.

Ensuite, la fonction utilise des expressions régulières pour vérifier si le mot de passe contient au moins une lettre majuscule, une lettre minuscule, un chiffre et un caractère spécial. Si l'un de ces tests échoue, la fonction renvoie false.

Enfin, si tous les tests réussissent, la fonction renvoie true.

Password confirmation :



```
const checkPasswordsMatch = (pwd, pwdConf) => {  
  if (pwd.value == pwdConf.value) {  
    return true;  
  } else {  
    return false;  
  }  
};
```

Cette fonction vérifie si les mots de passe entrés par l'utilisateur correspondent. Elle prend deux arguments : "pwd" et "pwdConf", qui sont censés être des objets représentant les champs de formulaire où l'utilisateur a saisi son mot de passe et sa confirmation de mot de passe.

La fonction utilise l'opérateur de comparaison "==" pour vérifier si la valeur du champ "pwd" est égale à la valeur du champ "pwdConf". Si c'est le cas, la fonction renvoie "true", sinon, elle renvoie "false".

Vérification des champs :

Pour ajouter de l'interactivité à la page d'inscription, nous utilisons les événements et le DOM pour permettre à l'utilisateur de recevoir des messages d'aide sous forme d'éléments HTML (p, span, etc.) lorsqu'il remplit le formulaire. Nous prenons en compte les règles de validation de notre code afin de permettre à l'utilisateur d'accéder à la page "index" s'il remplit le formulaire correctement.


```
const setFeedback = (isValid, element, msg) => {
  element.classList.toggle("is-valid", isValid);
  element.classList.toggle("is-invalid", !isValid);
  const parentNode = element.parentNode;
  const feedback = parentNode.getElementsByTagName("div")[0];
  feedback.classList.toggle("valid-feedback", isValid);
  feedback.classList.toggle("invalid-feedback", !isValid);
  feedback.innerHTML = isValid ? "Looks good !" : msg;
};
```

La fonction "setFeedback" est une fonction JavaScript qui permet de définir des messages de retour ("feedback") pour un élément HTML donné. Elle prend en entrée trois paramètres :

- "isValid" : un booléen indiquant si l'élément est valide ou non.
- "element" : l'élément HTML pour lequel le message de retour doit être défini.
- "msg" : le message de retour à afficher si l'élément n'est pas valide.

La fonction commence par ajouter ou supprimer des classes "is-valid" ou "is-invalid" à l'élément en fonction de la valeur de "isValid". Ces classes sont souvent utilisées en conjonction avec les classes Bootstrap "valid-" et "invalid-" pour afficher des styles différents en fonction de la validité de l'élément.

Ensuite, la fonction récupère le noeud parent de l'élément et utilise la méthode "getElementsByTagName" pour récupérer le premier élément de type "div" enfant de ce noeud parent. Cet élément de type "div" sera utilisé pour afficher le message de retour.

Enfin, la fonction ajoute ou supprime les classes "valid-feedback" ou "invalid-feedback" à l'élément "div" en fonction de la valeur de "isValid", et définit le contenu de cet élément en utilisant l'opérateur ternaire (? :) Pour afficher "Looks good !" si l'élément est valide ou le message d'erreur si l'élément n'est pas valide.

```

document.addEventListener("input", (e) => {
  switch (e.target) {
    case email:
      setFeedback(
        checkEmail(email),
        email,
        "Please enter a valid email address (example@example.ma)"
      );
      break;
    case username:
      setFeedback(
        checkUsername(username),
        username,
        "Invalid username (3 - 25 characters required)"
      );
      break;
    case pwd:
      setFeedback(
        checkPassword(pwd),
        pwd,
        "Invalid password (8+ characters,
        1 special character(!@#$$%^&*), 1 uppercase,
        1 lowercase, 1 digit required)"
      );
      break;
    case pwdConf:
      setFeedback(
        checkPasswordsMatch(pwd, pwdConf),
        pwdConf,
        "Please make sure that the confirmation password matches the original password"
      );
      break;
    case accept:

      accept.classList.toggle("is-valid", accept.checked);
      accept.classList.toggle("is-invalid", !accept.checked);
      break;

    default:
      alert("invalid request !")
      break;
  }
});

```

Ce code ajoute un gestionnaire d'événement "input" au document. Lorsque l'utilisateur saisit du texte dans n'importe quel élément du document, cet événement est déclenché et le gestionnaire d'événement est exécuté.

Le gestionnaire d'événement utilise un "switch" sur l'élément cible de l'événement (c'est-à-dire l'élément qui a déclenché l'événement). Selon l'élément cible, le gestionnaire d'événement appelle la fonction "setFeedback" avec différents arguments.

Par exemple, si l'élément cible est l'élément "email", le gestionnaire d'événement appelle la fonction "setFeedback" avec l'élément "email", le

résultat de la fonction "checkEmail" (qui vérifie si l'adresse e-mail saisie est valide ou non) et un message d'erreur spécifique. De même, si l'élément cible est l'élément "pwdConf", le gestionnaire d'événement appelle la fonction "setFeedback" avec l'élément "pwdConf", le résultat de la fonction "checkPasswordsMatch" (qui vérifie si le mot de passe et sa confirmation sont identiques) et un message d'erreur spécifique.

Le gestionnaire d'événement a également un cas "default" qui est exécuté si l'élément cible n'est pas l'un des éléments prévus dans le "switch". Dans ce cas, un message d'alerte est affiché.



```
submit.addEventListener("click", () => {
  if (
    !(checkUsername(username) &&
      checkEmail(email) &&
      checkPassword(pwd) &&
      checkPasswordsMatch(pwd, pwdConf) &&
      accept.checked)
  ) {
    if (!accept.checked) {
      accept
        .classList.remove("is-valid");
      accept
        .classList.add("is-invalid");

      alert("please accept our conditions and privacy policy !");
    } else {
      alert("Please complete the form before submitting");
    }
  } else {
    window.location.href = "../index.html";
  }
});
```

Ce code ajoute un gestionnaire d'événement "click" à l'élément "submit". Lorsque l'utilisateur clique sur cet élément, le gestionnaire d'événement est exécuté.

Le gestionnaire d'événement vérifie si tous les champs du formulaire sont valides en utilisant les fonctions de vérification "checkUsername", "checkEmail", "checkPassword" et "checkPasswordsMatch" et en vérifiant également si l'utilisateur a coché la case "accept" (qui correspond à un élément de type "checkbox").

Si tous les champs sont valides et que la case "accept" est cochée, le gestionnaire d'événement redirige l'utilisateur vers la page "index.html". Si l'un des champs n'est pas valide ou si la case "accept" n'est pas cochée, le gestionnaire d'événement affiche un message d'alerte indiquant à l'utilisateur ce qu'il doit faire pour soumettre le formulaire.

Navbar :

```
<div class="align-content-center w-100 pt-2 mb-5 shadow" id="navbar">

  <div class="row justify-content-between">
    <div class="col-1 d-flex justify-content-between ps-5">
      
      <span class="fw-bolder fs-5 ps-2 align-self-center pb-2" id="Logo" >ABDCourses </span>
    </div>
    <div class="col-3">
      <ul class="d-flex gap-3">
        <li id="home" class="btn btn-outline-primary rounded-pill "> home </li>
        <li id="coursesbtn" class="btn btn-primary rounded-pill "> Courses </li>
        <li id="about" class="btn btn-outline-primary rounded-pill "> about </li>
      </ul>
    </div>
  </div>
</div>
```

Ce code HTML représente une barre de navigation (navbar) qui comprend un logo et un menu avec trois liens.

La navbar est contenue dans un élément div avec les classes suivantes :

- **align-content-center** : centre verticalement le contenu de l'élément div.
- **W-100** : indique que l'élément div occupera toute la largeur de son parent.
- **Pt-2** : ajoute un espace de 2 points en haut de l'élément div.
- **Mb-5** : ajoute un espace de 5 points en bas de l'élément div.
- **shadow**: ajoute un effet d'ombre à l'élément div.

L'élément div a également un identifiant unique appelé **navbar**.

À l'intérieur de l'élément div de la navbar, il y a un élément div avec la classe **row** qui représente une ligne de colonnes Bootstrap. Cet élément div contient deux colonnes (éléments div avec la classe **col-1** et **col-3**), qui occupent respectivement 1/12 et 3/12 de la largeur de la ligne.

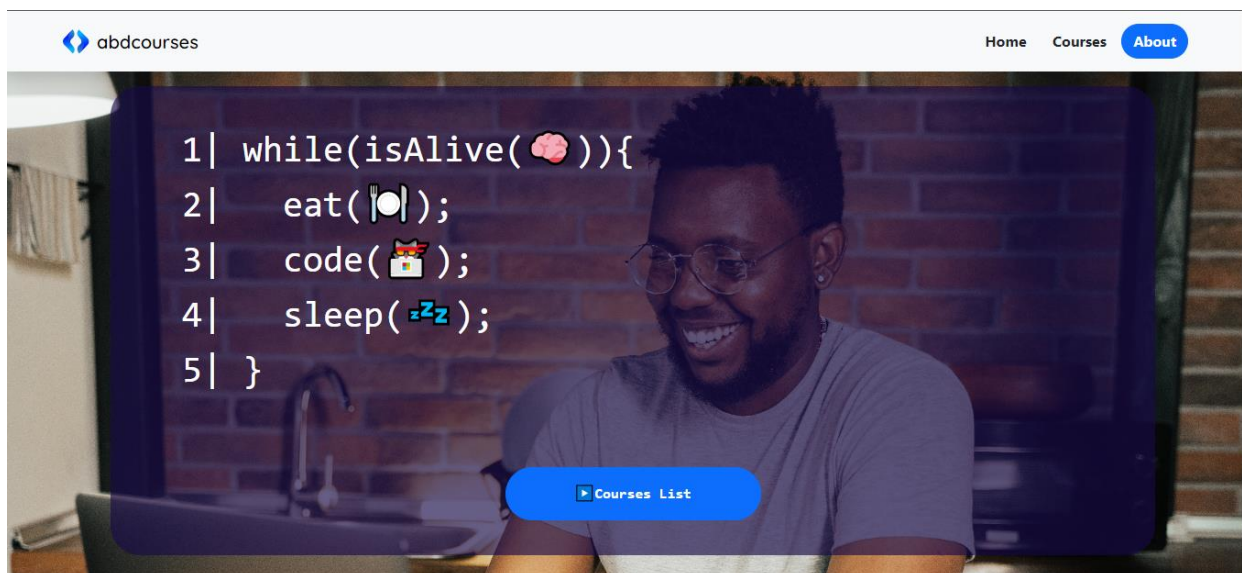
La première colonne contient un logo et un titre, tandis que la seconde colonne contient un menu avec trois liens (éléments li). Chaque lien est stylisé en

utilisant la classe **btn** de Bootstrap et les classes **rounded-pill** et **btn-outline-primary** ou **btn-primary** pour ajouter un style spécifique.

Le menu est contenu dans un élément **ul** avec la classe **d-flex** qui permet de disposer les liens de manière flexible. La classe **gap-3** ajoute un espace de 3 points entre chaque lien.

Chaque lien a également un identifiant unique qui correspond à son nom (**home**, **coursesbtn**, **about**). Ces identifiants peuvent être utilisés pour cibler et manipuler les liens en JavaScript.

Page index :



Partie html :

```
<div id="landing">
  <div class="container mt-3 rounded-5 d-flex flex-column justify-content-center align-
content-center" id="poster">
    <pre style=" font-family: Consolas, monaco, monospace;font-size:40px; color: white;" >
      1| while(isAlive(🐙)){
      2|   eat(🍌);
      3|   code(💻);
      4|   sleep(.2x);
      5| }
    </pre>
    <button style="font-family: Consolas, monaco, monospace; font-weight: bolder;"
id="coursesbtn" class="btn btn-primary w-25 p-3 rounded-pill align-self-center">▶Courses List</button>
  </div>
</div>
<h1 class="align-self-center mt-3"><span style="color: rgb(238, 215, 8);">&lt; /</span>
courses<span style="color: rgb(238, 215, 8);">&gt;</span></h1>
<div class="align-self-center mb-5" id="randCourses" >

</div>
<button id="coursesbtn" class="btn btn-primary p-2 rounded-pill align-self-center mb-5 "
style="width: 200px;">all courses</button>
<div class="container text-center">
  <p><span class="fw-bolder"> abdcourses</span> &#169; 2022 - 2023</p>
</div>
```

Ce code HTML représente une page Web avec un écran de bienvenue et une liste de cours.

L'écran de bienvenue est représenté par un élément div avec l'identifiant **landing** qui contient un élément div avec l'identifiant **poster**. L'élément **poster** contient du code préformaté (élément **pre**) avec du style appliqué pour le mettre en valeur et un bouton (élément **button**) avec l'identifiant **coursesbtn** qui permet à l'utilisateur de naviguer vers la liste de cours.

Le titre de la page (<**h1**>) est suivi d'un élément div avec l'identifiant **randCourses** qui sert de conteneur pour afficher une liste de cours aléatoires.

Il y a également un autre bouton (élément **button**) avec l'identifiant **coursesbtn** qui permet à l'utilisateur de naviguer vers la liste complète des cours.

Enfin, il y a un élément div avec la classe **container** de Bootstrap qui contient un paragraphe (élément **p**) avec un copyright et une année.

Le code HTML utilise des classes de Bootstrap et des styles en ligne pour ajouter un style et une mise en forme spécifiques aux éléments de la page. Il utilise également des caractères de code spéciaux pour ajouter de l'humour et de la personnalité à la page.



< / courses >

 <p>\$64 Intro to JavaScript</p>	 <p>\$80 JavaScript Fundamentals</p>	 <p>\$54 Intro to css</p>
---	---	--

abdcourses © 2022 - 2023

Partie interactivité :

List des cours :

Pour afficher les cours d'une manière aléatoire, nous créons d'abord un tableau qui contient des objets de la forme suivante :

```
let courses = [
  {
    title: "un titre",
    img: "le/chemin/de/l/image",
    category: "categorie du lesson",
    price: 999 // ici le prix
  },
  ...
]
```

Générer des cours aléatoirement :

Nous avons ensuite créé une fonction qui retourne un tableau contenant 3 éléments choisis aléatoirement :

```
function getRandomCourses(listOfCourses) {
  var randomCourses = [];
  while (randomCourses.length < 3) {
    var index = Math.floor(Math.random() * listOfCourses.length);
    var course = listOfCourses[index];
    if (!randomCourses.find( randomCourse => randomCourse.title ==course.title &&
                                randomCourse.price ==course.price)) {
      randomCourses.push(course);
    }
  }
  return randomCourses;
}
```

Cette fonction prend en entrée une liste de cours (**listOfCourses**) et retourne une liste de trois cours aléatoires tirés de cette liste.

La fonction commence par déclarer une liste vide appelée **randomCourses** qui va contenir les cours aléatoires qui seront retournés à la fin de la fonction.

Ensuite, la fonction utilise une boucle **while** qui s'exécutera tant que la longueur de **randomCourses** est inférieure à 3. À chaque itération de la boucle, la fonction génère un nombre aléatoire compris entre 0 et la longueur de **listOfCourses** (exclu) et utilise ce nombre comme index pour récupérer un élément de **listOfCourses**. Cet élément est stocké dans la variable **course**.

La fonction vérifie ensuite si un cours ayant le même titre et le même prix que **course** est déjà présent dans **randomCourses** en utilisant la méthode **find()**. Si **find()** retourne **false**, cela signifie que **course** n'est pas déjà présent dans **randomCourses**, et le cours est ajouté à cette liste en utilisant la méthode **push()**.

Une fois que la boucle **while** s'est terminée, la fonction retourne la liste de cours aléatoires **randomCourses**.

Affichage de cours :

Et pour afficher ces Lessons sous forma des cartes :

```
const DisplayCards = (parentElement, content) => {
  let newContent = content.map((element) => {
    const htmlContent = `
<div class="card" style="width: 100%;">
  
  <div class="card-body">
    <h7 class="card-title fw-bold text-success">${element.price}</h7>
    <br>
    <h7 style="font-family: 'Unbounded', cursive" class="card-title">${element.title}</h7>
  </div>
</div>
`;
    return htmlContent;
  });
  newContent = newContent.join("");
  parentElement.innerHTML = newContent;
};
```

Cette fonction prend en entrée un élément DOM parent (**parentElement**) et un tableau de contenu (**content**) et affiche chaque élément du tableau sous forme de cartes dans l'élément DOM parent.

La fonction utilise la méthode **map()** pour parcourir chaque élément de **content** et générer du code HTML pour chaque carte. Cette fonction de flèche est

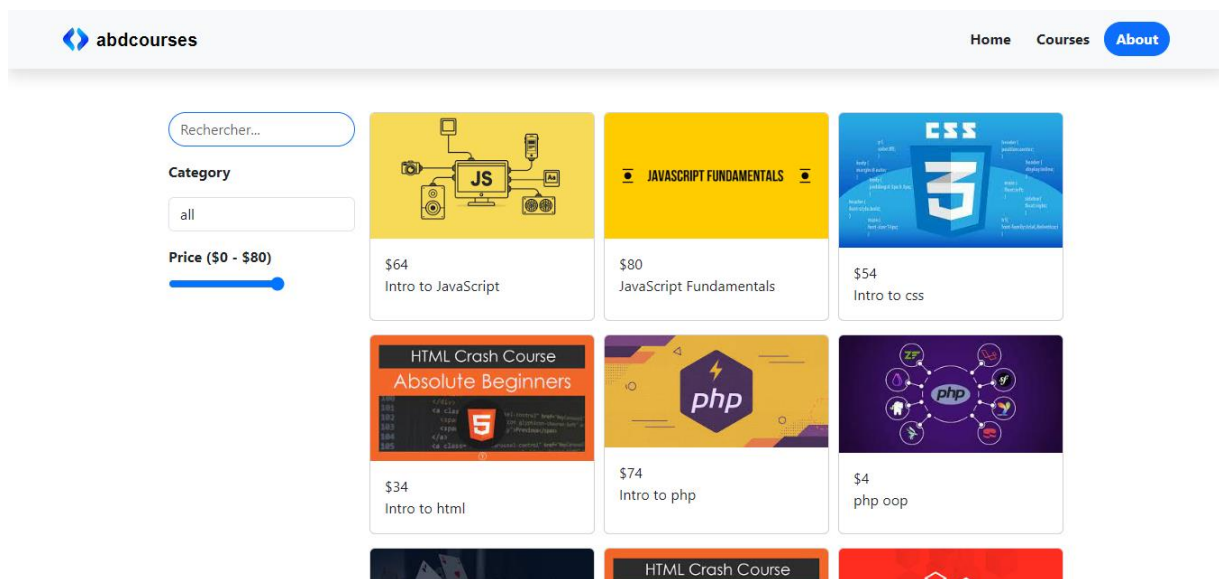
utilisée pour transformer chaque élément de **content** en une chaîne de caractères HTML qui représente une carte.

Le code HTML généré inclut des valeurs de chaque élément de **content**, telles que l'image, le prix et le titre, qui sont insérées dans le code HTML en utilisant des interpolations de chaînes.

Une fois que la méthode **map** a terminé son exécution, la fonction utilise la méthode **join()** pour convertir le tableau généré en une chaîne de caractères (pour éviter des problèmes des virgules lors du déplacement de ces cours vers notre page) et stocke le résultat dans la variable **newContent**.

Enfin, la fonction utilise la propriété **innerHTML** de l'élément DOM parent pour remplacer son contenu actuel par le contenu généré. Cela affichera les cartes dans l'élément DOM parent.

Courses page :



Filtrage :

```

<div class="container d-flex justify-content-center gap-3 p">
  <div id="filter">
    <div class="form-group">
      <input id="search" type="text" class="form-control rounded-pill border-primary"
placeholder="Rechercher...">
    </div>
    <div class="form-group">
      <label for="category" class="fw-bold mt-3 mb-3">Category</label>
      <select class="form-control" id="category">
        <option selected>all</option>
        <option value="js">javascript</option>
        <option value="css">css</option>
        <option value="html">html</option>
        <option value="php">php</option>
      </select>
    </div>

    <div class="form-group">
      <label for="customRange3" class="form-label fw-bold mt-3">Price (<span id="displayRange"></span>)</label>
      <br>
      <input type="range" class="costume-range" value="80" min="0" max="80" step="1" id="customRange3">
    </div>

  </div>
  <div id="courseList"></div>
</div>

```

Ce code HTML représente un filtre de recherche de cours comprenant une barre de recherche, un menu déroulant de catégories et une glissière de prix.

Le filtre est contenu dans un élément div avec la classe **container** de Bootstrap, qui permet de centrer et de limiter la largeur du contenu. L'élément div a également la classe **d-flex** qui permet de disposer les éléments enfants de manière flexible, et la classe **gap-3** qui ajoute un espace de 3 points entre chaque élément enfant.

À l'intérieur de l'élément div du filtre, il y a un élément div avec l'identifiant **filter** qui contient les trois éléments de filtrage (barre de recherche, menu déroulant de catégories et glissière de prix).

La barre de recherche est contenue dans un élément div avec la classe **form-group** de Bootstrap et est représentée par un champ de saisie (élément input) avec la classe **form-control** et la classe **rounded-pill** qui ajoute un style arrondi aux coins du champ de saisie. Le champ de saisie a également l'identifiant **search** qui peut être utilisé pour cibler et manipuler le champ de saisie en JavaScript.

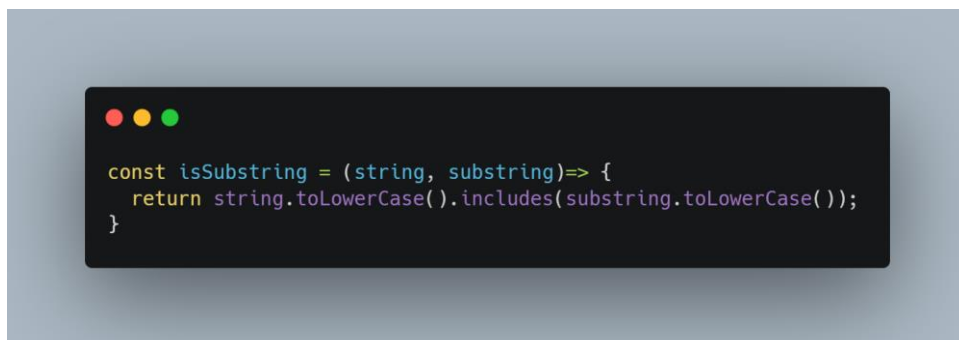
Le menu déroulant de catégories est également contenu dans un élément div avec la classe **form-group** et est représenté par un élément select avec la classe **form-control**. L'élément select contient plusieurs options (éléments option) qui représentent les différentes catégories de cours disponibles.

La glissière de prix est contenue dans un élément div avec la classe **form-group** et est représentée par un élément input de type "range" avec la classe **costume-range**. L'élément input a des attributs qui définissent les valeurs minimale et maximale de la plage de prix et l'intervalle entre chaque étape (**min**, **max** et **step**). L'élément input a également un identifiant unique appelé **customRange3** qui peut être utilisé pour cibler et manipuler la glissière de prix en JavaScript.

Il y a également un élément span avec l'identifiant **displayRange** qui est utilisé pour afficher la valeur actuelle de la plage de prix.

Enfin, il y a un élément div avec l'identifiant **courseList** qui sert de conteneur pour afficher la liste des cours filtrés.

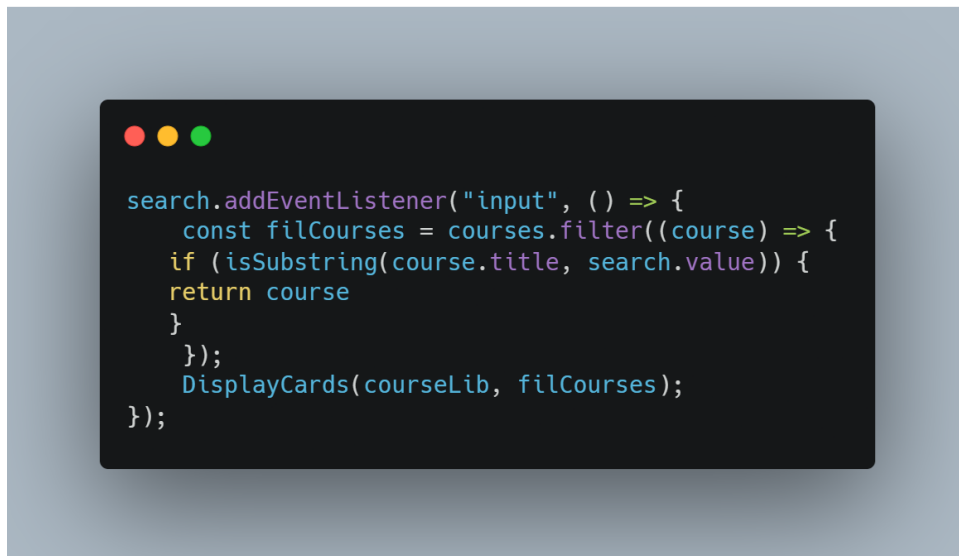
Le filtre de recherche de cours peut être utilisé pour rechercher et filtrer les cours disponibles en fonction de différents critères, tels que le titre, la catégorie et le prix. Les éléments de filtrage peuvent être liés à des fonctions JavaScript qui sont exécutées lorsque l'utilisateur interagit avec eux (par exemple, en saisissant du texte dans la barre de recherche ou en sélectionnant une catégorie dans le menu déroulant). Ces fonctions peuvent être utilisées pour mettre à jour la liste des cours affichés en fonction des critères de filtrage choisis par l'utilisateur.



La fonction "isSubstring" prend en entrée une chaîne de caractères "string" et une autre chaîne de caractères "substring", et renvoie "true" si "substring" est incluse dans "string", ou "false" sinon.

La fonction utilise la méthode "toLowerCase" de l'objet "String" pour convertir "string" et "substring" en minuscules. Elle utilise ensuite la méthode "includes" de l'objet "String" pour vérifier si "substring" est incluse dans "string". La méthode "includes" renvoie "true" si la chaîne de caractères cible est trouvée, ou "false" sinon.

La fonction "isSubstring" est insensible à la casse, c'est-à-dire qu'elle renvoie "true" si "substring" est incluse dans "string" quelle que soit la casse de ces chaînes de caractères.



La fonction **search.addEventListener** ajoute un événement **input** à l'élément HTML **search**. Lorsque l'événement "input" est déclenché (par exemple, lorsque l'utilisateur tape du texte dans le champ de recherche), la fonction anonyme associée à l'événement est exécutée.

La fonction anonyme commence par déclarer une variable "**filCourses**" qui contient le résultat de l'appel de la méthode "filter" sur "courses". La méthode "filter" crée un nouveau tableau qui ne contient que les éléments du tableau d'origine pour lesquels la fonction de filtre renvoie "**true**". Dans ce cas, la fonction de filtre est une fonction anonyme qui prend en entrée un élément "course" et qui vérifie si la chaîne de caractères "**search.value**" est incluse dans le titre du cours (grâce à l'appel de la fonction **isSubstring()**) et enfin on affiche résultat dans notre page à l'aide de la fonction qu'on a vue précédemment.

```
budgetRange.addEventListener("input", () => {
  displayRange.innerHTML = "0 - " + budgetRange.value;

  const budgetCourses = courses.filter((course) => {
    return course.price < budgetRange.value;
  });
  DisplayCards(courseLib, budgetCourses);
});
```

```
category.addEventListener("input", () => {
  if (category.value !== "all") {
    const catCourses = courses.filter((course) => {
      return course.category === category.value;
    });
    DisplayCards(courseLib, catCourses);
  } else {
    DisplayCards(courseLib, courses);
  }
});
```

La fonction "category.addEventListener" ajoute un événement "input" à l'élément HTML "category". Lorsque l'événement "input" est déclenché (par exemple, lorsque l'utilisateur sélectionne une catégorie dans le menu déroulant), la fonction anonyme associée à l'événement est exécutée.

La fonction anonyme vérifie si la valeur de "category.value" est égale à "all". Si c'est le cas, elle appelle la fonction "DisplayCards" avec les arguments "courseLib" et "courses". Cela permet d'afficher tous les cours dans la bibliothèque de cours (courseLib c'est l'emplacement).

Si "category.value" est différent de "all", la fonction anonyme déclare une variable "catCourses" qui contient le résultat de l'appel de la méthode "filter" sur "courses". La méthode "filter" crée un nouveau tableau qui ne contient que les éléments de la catégorie correspondant à la valeur du "category.value" .

Affichage :



```
window.addEventListener("DOMContentLoaded", () => {  
  DisplayCards(courseLib, courses);  
});
```

Cette ligne de code utilise l'événement **DOMContentLoaded** de la fenêtre (**window**) pour exécuter une fonction anonyme une fois que le DOM (Document Object Model) de la page a été chargé et parsé.

La fonction anonyme appelle une fonction nommée **DisplayCards** en lui passant deux arguments : **courseLib** et **courses**. **courseLib** et **courses** sont des variables qui sont définies ailleurs dans le code et qui contiennent respectivement un élément DOM parent et un tableau de contenu.

La fonction **DisplayCards** est censée afficher le contenu du tableau **courses** sous forme de cartes dans l'élément DOM parent **courseLib**.

L'événement **DOMContentLoaded** est utile lorsque vous souhaitez exécuter du code une fois que le DOM est prêt, mais avant que les images, les feuilles de style et les scripts externes soient entièrement chargés. Cela permet d'éviter que le code ne s'exécute avant que le DOM soit prêt, ce qui pourrait entraîner des erreurs ou des comportements indésirables.

Page about :



Notre projet consiste à créer un site web de vente de leçons de développement web. Il s'adressera aux développeurs débutants et intermédiaires souhaitant acquérir de nouvelles compétences et se perfectionner dans le développement web. Le site proposera une variété de leçons sur les technologies et les langages de développement web les plus populaires, tels que HTML, CSS, JavaScript, PHP, etc. Les utilisateurs pourront s'inscrire, sélectionner les leçons qui les intéressent. Ils peuvent aussi utiliser notre système de filtration des leçons par prix, catégorie et langage de programmation pour trouver celles qui correspondent le mieux à vos besoins et à votre budget. Notre objectif est de créer un site web convivial, intuitif et performant, qui réponde aux besoins et aux attentes des utilisateurs en matière de "développement web".

Il est important de créer une page "about" pour expliquer l'objectif de celle-ci aux utilisateurs. Cela peut aider à établir une relation de confiance avec vos visiteurs et à leur donner une idée de ce à quoi ils peuvent s'attendre en utilisant votre site. Vous pouvez également utiliser cette page pour présenter l'équipe derrière votre plateforme, présenter votre entreprise et partager toute information supplémentaire que vous souhaitez partager avec vos utilisateurs. Assurez-vous de mettre à jour régulièrement cette page pour que les utilisateurs puissent rester informés de l'évolution de votre plateforme et de ses objectifs.

Conclusion :

Faire ce mini projet sur JavaScript nous a permis de mettre en pratique nos connaissances et de nous familiariser davantage avec le langage de programmation. Nous avons appris de nouvelles techniques et concepts, ainsi que des façons de résoudre des problèmes spécifiques à JavaScript. En travaillant sur ce projet, nous avons découvert de nouvelles idées ou des domaines à explorer davantage.

En conclusion, faire ce mini projet sur JavaScript a été bénéfique pour notre apprentissage et notre compréhension du langage, et nous a permis de développer nos compétences en programmation. Cela nous a aidés à être plus efficaces et à réussir dans nos projets futurs.

Bibliographie :

- MDN Web Docs (<https://developer.mozilla.org/en-US/docs/Web>) : une référence en ligne complète pour le développement Web, avec des tutoriels et des exemples de code pour JavaScript, CSS et HTML.

- W3Schools (<https://www.w3schools.com/>) : un site qui propose des tutoriels interactifs et des exemples de code pour JavaScript, CSS et HTML, ainsi que pour de nombreuses autres technologies Web.
- Codecademy (<https://www.codecademy.com/>) : un site qui propose des cours en ligne et des exercices pratiques pour apprendre à coder en JavaScript, CSS et HTML, ainsi que dans d'autres langages de programmation.
- freeCodeCamp (<https://www.freecodecamp.org/>) : un site qui propose des cours en ligne et des projets pratiques pour apprendre à coder en JavaScript, CSS et HTML, ainsi que dans d'autres langages de programmation.
- Tutorial Et les exemples Proposer par prof Mohammed AMESKA.