LE FORMULAIRE EN JAVASCRIPT

I) LA VALIDATION DU FORMULAIRE

1) Création du formulaire : HTML

Dans un premier temps nous allons devoir créer un formulaire en HTML ET mettre nos id et nos class afin de pouvoir faire le design et les fonctionnalités par la suite

Je ne reviendrai pas sur la structure du formulaire

Voici le code HTML le formulaire :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <form id="signup" class="form">
            <h1>S'enregistrer</h1>
            <div class="form-field error success">
                <label for="username">Nom : </label>
                <input type="text" name="username" id="username"
autocomplete="off">
                <small></small>
            </div>

            <div class="form-field error success">
                <label for="email">Email:</label>
                <input type="text" name="email" id="email"
autocomplete="off">
                <small></small>
            </div>

            <div class="form-field error success">
```

```
                <label for="password">Mot de passe</label>
                <input type="password" name="password" id="password"
autocomplete="off">
                <small></small>
            </div>


            <div class="form-field error success">
                <label for="confirm-password">Confirmation du mot de
passe</label>
                <input type="password" name="confirm-password"
id="confirm-password" autocomplete="off">
                <small></small>
            </div>

            <div class="form-field error success">
                <input type="submit" value="Sign Up">
            </div>
        </form>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

ATTENTION À BIEN LIER VOTRE CSS ET VOTRE JAVASCRIPT !!!!

Nous allons donc passer au CSS

2) LE CSS

Nous allons embellir notre formulaire de façon à ce qu'il soit clair et épuré. Je ne reviendrai pas non plus sur le CSS.

Voici le code CSS :

```css
@import
url('https://fonts.googleapis.com/css?family=Open+Sans&display=swap');

:root {
    --error-color: #dc3545;
    --success-color: #28a745;
    --warning-color: #ffc107;
}

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}


body {
    font-family: 'Open Sans', sans-serif;
    font-size: 16px;
    background-color: #f4f4f4;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    margin: 0;
}

.container {
    background-color: #fff;
    padding: 1em;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);
    width: 400px;
}
```

```
.form {
    padding: 10px 20px;
}

.form h1 {
    font-size: 1.5em;
    text-align: center;
    margin-bottom: 20px;

}

.form-field {
    margin-bottom: 5px;

}

.form-field label {
    display: block;
    color: #777;
    margin-bottom: 5px;
}

.form-field input {
    border: solid 2px #f0f0f0;
    border-radius: 3px;
    padding: 10px;
    margin-bottom: 5px;
    font-size: 14px;
    display: block;
    width: 100%;
}

.form-field input:focus {
    outline: none;
}

.form-field.error input {
    border-color: var(--error-color);
}

.form-field.success input {
    border-color: var(--success-color);
```

```
}


.form-field small {
    color: var(--error-color);
}


/* button */
.btn {
    width: 100%;
    padding: 3%;
    background: #007bff;
    border-bottom: 2px solid #007bff;
    border-top-style: none;
    border-right-style: none;
    border-left-style: none;
    color: #fff;
    text-transform: uppercase;
}

.btn:hover {
    background: #0069d9;
    cursor: pointer;
}

.btn:focus {
    outline: none;
}
```

Maintenant que notre formulaire est plutôt agréable nous allons nous attarder sur le Javascript du formulaire

3) LE JAVASCRIPT

Le but de l'exercice ici est de faire la validation du formulaire.

a) les premières constantes

Nous allons donc dans un premier temps récupérer les champs du formulaire dans des constantes ainsi que le formulaire .

Nous allons donc utiliser ‘document.querySelector’ et comme dans notre HTML nous avons défini nos inputs avec des id on dira que l’on veut les id de la façon suivante :

```
const toto = document.querySelector('#nomdelid');
```

et donc pour le formulaire en question nos variables sont :

```
const usernameEl = document.querySelector('#username');
const emailEl = document.querySelector('#email');
const passwordEl = document.querySelector('#password');
const confirmPasswordEl = document.querySelector('#confirm-password');

const form = document.querySelector('#signup');
```

Ici nous récupérons tous nos champs et le formulaire par la même occasion. Si nous revenons sur le html nous pouvons voir nos différents id.

```
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <form id="signup" class="form">
            <h1>S'enregistrer</h1>
            <div class="form-field error success">
                <label for="username">Nom : </label>
                <input type="text" name="username" id="username" autocomplete="off">
                <small></small>
            </div>

            <div class="form-field error success">
                <label for="email">Email:</label>
                <input type="text" name="email" id="email" autocomplete="off">
                <small></small>
            </div>

            <div class="form-field error success">
                <label for="password">Mot de passe</label>
                <input type="password" name="password" id="password" autocomplete="off">
                <small></small>
            </div>
```

Voila comment nous avons pu faire notre lien pour récupérer chaque champs

b) La fonction fléchée

Ensuite nous allons devoir vérifier si nos champs sont vides ou non.
Nous allons voir donc la fonction fléchée.

Pour faire notre vérification le code sera le suivant :

```
// constante pour vérifier si le champ est vide ou non
const isRequired = value => value === '' ? false : true;
```

ici nous avons **“const”** qui détermine que c’est une constante.
Elle s’appelle isRequired qui est = à la value (du champ)
Ensuite nous avons une partie de code qui est la suivante :

```
=> value === '' ? false : true;
```

cela est une condition et remplace notre **if(){}else{}**
Donc si nous traduisons cela :

si notre value est strictement égal à un ensemble vide alors ça retourne false sinon ça retourne true

Le **“?”** est un si et le **“:”** est alors.

Nous allons faire de même pour dire que le champ doit être compris entre n caractères et n caractères

Pour cela nous allons déclarer notre fonction fléchée comme la précédente avec :

```
const isBetween =(length,min,max)=>length<min || length>max ?
false : true;
```

Notre fonction fléchée s’appelle isBetween il nous faut en paramètre la longueur, un minimum et un maximum si la longueur est inférieure au minimum ou si la longueur est supérieur au maximum ça nous retourne false sinon ça nous retourne true .

c) *Vérif de l’email*

Nous allons devoir vérifier l’email et toujours dans une fonction fléchée, nous allons devoir utiliser une regex.

La regex est appelée expression régulière elle nous permet de valider ou non un format par exemple ici le mail.

Nous savons qu’un mail a le caractère **“@”** et aussi un **“.” ,** nous allons donc vérifier ce format. (pour plus d’informations sur les regex voir le cours ou les regex en cliquant ici).
Par conséquent dans notre fonction fléchée nous avons ceci.

```
const isEmailValid = (email) => {
        const re =
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-
Z]{2,}))$/;
        return re.test(email);
    };
```

une const que l'on nom re pour regex et nous testons donc la regex sur l'email car en paramètre nous reprenons l'email.

d) Vérification du mot de passe

Même principe avec le mot de passe sauf ici nous voulons un mot de passe d'une longueur de 8 caractères avec majusucule minuscule chiffre et caractère spéciaux définis.

En paramètre de notre fonction fléchée nous avons donc password et la fonction nous retourne donc si le test du mot de passe comme ceci :

```
const isPasswordSecure = (password) => {
        const re = new RegExp("^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-
9])(?=.*[!@#\$%\^&\*])(?=.{8,})");
        return re.test(password);
    };
```

e) Afficher les erreurs

Maintenant nous voulons que les erreurs soient affichées lorsque l'on a un champ qui n'est pas validé.

Toujours dans une fonction fléchée que nous définissons showError, en paramètre nous avons input et message car nous allons cibler l'input et afficher le message en question

Pour le moment nous avons donc ceci :

```
const showError = (input, message)=>{

}
```

Nous allons dans notre fonction fléchée reprendre le form-field element afin de pouvoir afficher le css de la class **error**

```
// reprendre le form-field element
        const formField = input.parentElement;
```

Ensuite nous allons ajouter la class error avec add() et supprimer la class success avec remove() comme ceci :

```
// ajouter la class error et supprimer la class success
        formField.classList.remove('success');
        formField.classList.add('error');
```

Bien évidemment on va rechercher parmi les class de notre css avec classList.

Pour finir nous allons vouloir afficher le message d'error dans la balise **<small></small>** comme ci-dessou :

```
// afficher le message d'erreur
        const error = formField.querySelector('small');
        error.textContent = message;
```

Nous avons donc au final notre fonction showError comme ceci :

```
const showError = (input, message) => {
        // reprendre le form-field element
        const formField = input.parentElement;
        // ajouter la class error et supprimer la class success
        formField.classList.remove('success');
        formField.classList.add('error');

        // afficher le message d'erreur
        const error = formField.querySelector('small');
        error.textContent = message;
    };
```

f) *Afficher lorsque le formulaire est ok*

Nous allons faire la même chose mais dans le sens inverse c'est à dire quand les champs sont valides.

Attention le message sera vide lorsque les champs sont valides.

Je ne le détaille pas puisque c'est semblablement la même chose sauf :

```
const showSuccess = (input) => {
        // reprendre le form-field element
        const formField = input.parentElement;

        // ajouter la class success et supprimer la class error
        formField.classList.remove('error');
        formField.classList.add('success');

        // cacher le message d'erreur
        const error = formField.querySelector('small');
        error.textContent = '';
    }
```

Oui j'ai dit sauf car nous n'avons en paramètre que l'input et au niveau du message on met un ensemble vide :
error.textContent = ' ';

g) Validation du nom

Maintenant que toutes nos fonctions sont faites nous allons pouvoir les appliquer dans d'autres fonctions.

Nous allons tout d'abord nous occuper de la validation du nom et pour cela nous aurons besoin des fonctions créées en amont comme :

- isRequired
- showError
- isBetween
- showSucces

Toujours dans une fonction fléchée que nous allons appeler checkUsername sans paramètres :

```
const checkUsername=()=>{

}
```

Nous allons créer une variable valid que nous passons à false :

```
let valid = false;
```

Nous faisons deux constantes :

la première min = 3

la deuxième max= 25

En procédant ainsi je veux qu'il y'ai au minimum 3 caractères et pas plus de 25.

```
let valid = false;
        const min = 3,
            max = 25;
```

Nous allons utiliser également le trim() qui nous servira d'enlever les espaces au cas où un utilisateur aurait mis un espace avant d'écrire afin de compter réellement les vrais caractères et que l'espace ne soit pas compté comme caractère.

Nous faisons donc une constante username qui reprendra la constante du départ de l'input soit usernameEl on y prend donc sa value et on y fait le trim sur la value que l'on reprend comme ci-dessous :

```
const username = usernameEl.value.trim();
```

Ensuite nos conditions comme ceci :

```
if (!isRequired(username)) {
            showError(usernameEl, 'Le champ ne peut être vide');
        } else if (!isBetween(username.length, min, max)) {
            showError(usernameEl, `le nom doit être compris entre ${min} et
${max} caractères.`)
        } else {
            showSuccess(usernameEl);
            valid = true;
        }
        return valid;
```

première condition si le champ est vide donc on afficher le message d'erreur en question en appelant la fonction **showError()** sinon si le userName n'est pas dans la taille souhaitée de même on affiche l'erreur en appelant la fonction **showError()** et sinon valid passe à true et on utilise la fonction **showSuccess()**

Attention le trim est une très bonne pratique

Le rendu de la fonction de la validation du nom est comme ceci :

```
    //validation du nom
const checkUsername = () => {

        let valid = false;
        const min = 3,
            max = 25;
        const username = usernameEl.value.trim();

        if (!isRequired(username)) {
            showError(usernameEl, 'Le champ ne peut être vide');
        } else if (!isBetween(username.length, min, max)) {
            showError(usernameEl, `le nom doit être compris entre ${min} et
${max} caractères.`)
        } else {
            showSuccess(usernameEl);
            valid = true;
        }
        return valid;
    }
```

h) Validation du mail

Faisons de même avec le mail :

```
const checkEmail = () => {
        let valid = false;
        const email = emailEl.value.trim();
        if (!isRequired(email)) {
            showError(emailEl, 'le champ ne peut être vide');
        } else if (!isEmailValid(email)) {
            showError(emailEl, "L'adresse mail ne peut être valide")
        } else {
            showSuccess(emailEl);
            valid = true;
        }
        return valid;
    }
```

Ici nous avons comme fonction fléchée checkMail, à l'intérieur de cette fonction nous avons la variable valid que l'on passe à nouveau à false.
Nous faisons le trim() sur le mail.

Ensuite nous avons nos conditions à gérer, avec les messages d'erreur (showError) et le success avec showSuccess et le valid qui passe à true

i) Validation du mot de passe

La même chose pour le mot de passe :

```
//validation du mot de passe
const checkPassword = () => {

        let valid = false;

        const password = passwordEl.value.trim();

        if (!isRequired(password)) {
            showError(passwordEl, 'le mot de passe ne peut être vide');
        } else if (!isPasswordSecure(password)) {
            showError(passwordEl, 'Le mot de passe doit avoir au moins 8 caractères, il doit comporter une minuscule,une majuscule, un chiffre et un caractère spécial parmi les suivants (!@#$%^&*)');
        } else {
            showSuccess(passwordEl);
            valid = true;
        }

        return valid;
    };
```

Toujours le même principe sur les variable constantes et conditions pour la vérification

j) Vérification de la confirmation du mot de passe

Ce sera le même principe sauf que nous rajouterons une contante que l'on va appeler password et nous allons reprendre la value puis y faire un trim en plus de la constante confirmPassword

Dans nos conditions nous regardons si le champ est vide ou non puis nous vérifions si le mot de passe et sa confirmation sont différents. Si c'est le cas ça nous renvoie un message d'erreur avec la fonction **showError()**, sinon si tout est ok c'est valid et on met en place la fonction **showSucces()**

```
   //validation de la confirmation du mot de passe
const checkConfirmPassword = () => {
        let valid = false;

        const confirmPassword = confirmPasswordEl.value.trim();
        const password = passwordEl.value.trim();

        if (!isRequired(confirmPassword)) {
            showError(confirmPasswordEl, 'Entrez votre mot de passe');
        } else if (password !== confirmPassword) {
            showError(confirmPasswordEl, "Votre mot de passe et la
confirmation n'est pas bonne");
        } else {
            showSuccess(confirmPasswordEl);
            valid = true;
        }

        return valid;
    };
```

k) Soumettre le formulaire

Tout d'abord nous allons reprendre notre constante form pour reprendre le formulaire en entier puis nous allons utiliser le addEventListener et nous allons y faire un submit comme ci-dessous :

```
form.addEventListener('submit',function(e){
```

```
});
```

Ensuite, nous allons utiliser le event.preventDefault qui nous empêchera de soumettre le formulaire si nous avons un champ qui n'est pas vérifier.

```
// utilisation du prevent Default
    e.preventDefault();
```

Puis nous allons faire des variables :

```
 //validation des champs
    let isUsernameValid = checkUsername();
    let isEmailValid = checkEmail();
    let isPasswordValid = checkPassword();
    let isConfirmPasswordValid = checkConfirmPassword();
```

Ces variable comme nous pouvons le voir reprennent les fonctions de validation créées en amont

Nous allons créer une dernière variable que l'on va appeler isFormValid et cette variable va englober la totalité des validations faites juste avant.

```
  let isFormValid = isUsernameValid &&
        isEmailValid &&
        isPasswordValid &&
        isConfirmPasswordValid;
```

et ensuite on va soumettre le formulaire de cette façon :

```
 // Soumettre le formulaire
    if (isFormValid==false) {
        e.preventDefault();
        alert("L'envoie a échoué")
    }else{
        alert("Bravo l'envoie du formulaire a été effectué")
    }
```

Donc ici si notre formulaire n'est pas valide on utilise le e.preventDefault pour annuler l'envoie et une alert nous informe que l'envoie du formulaire a échoué.
Et si le formulaire d'envoie est ok alors une alert nous renvoie que le formulaire a été envoyé

Votre formulaire fonctionne avec les messages d'erreur qui s'affiche correctement.