# LES FORMULAIRES EN JAVASCRIPT

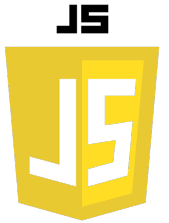
Les formulaires sont une partie essentielle de toute application web. Ils sont utilisés pour collecter des données de l'utilisateur, comme des informations de connexion, des données de profil, des informations de paiement, etc.

**Structure de base d'un formulaire HTML**
Avant de plonger dans le JavaScript, il est important de comprendre la structure de base d'un formulaire HTML. Voici un exemple simple :

```
<form id="myForm">
  <label for="name">Nom:</label><br>
  <input type="text" id="name" name="name"><br>
  <label for="email">Email:</label><br>
  <input type="text" id="email" name="email"><br>
  <input type="submit" value="Submit">
</form>
```

Dans cet exemple, nous avons un formulaire avec deux champs : un pour le nom et un pour l'email. Chaque champ a un label associé, et le formulaire a un bouton submit.

# LES FORMULAIRES EN JAVASCRIPT

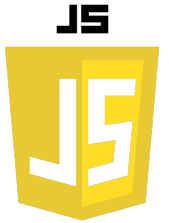



**Accéder aux données du formulaire en JavaScript :**
Pour accéder aux données du formulaire en JavaScript, nous pouvons utiliser l'objet document pour sélectionner le formulaire et ses éléments. Par exemple :

```
var form = document.getElementById('myForm');
var name = form.elements['name'].value;
var email = form.elements['email'].value;
```

Dans cet exemple, nous utilisons document.getElementById pour sélectionner le formulaire, puis nous utilisons la propriété elements pour accéder aux champs du formulaire.

# LES FORMULAIRES EN JAVASCRIPT




**Gérer la soumission du formulaire**
Lorsque l'utilisateur soumet le formulaire, nous pouvons utiliser JavaScript pour gérer l'événement de soumission et faire quelque chose avec les données du formulaire. Par exemple :

```
form.addEventListener('submit', function(event) {
  event.preventDefault(); // Empêche la soumission par défaut du formulaire
  var name = form.elements['name'].value;
  var email = form.elements['email'].value;
  console.log('Name: ' + name + ', Email: ' + email);
});
```

Dans cet exemple, nous ajoutons un écouteur d'événements submit au formulaire. Lorsque le formulaire est soumis, nous empêchons la soumission par défaut (qui rafraîchirait la page), puis nous récupérons les données du formulaire et les affichons dans la console.

**Validation côté client**

La validation côté client est effectuée dans le navigateur de l'utilisateur avant que les données ne soient envoyées au serveur. Elle permet d'améliorer l'expérience utilisateur en fournissant des retours immédiats et en évitant un aller-retour inutile au serveur.

Voici un exemple simple de validation côté client en JavaScript :

```
form.addEventListener('submit', function(event) {
  event.preventDefault(); // Empêche la soumission par défaut du formulaire

  var name = form.elements['name'].value;
  var email = form.elements['email'].value;

  if (name === '') {
    alert('Le nom ne peut pas être vide');
    return;
  }

  if (email === '') {
    alert('L\'email ne peut pas être vide');
    return;
  }

  console.log('Name: ' + name + ', Email: ' + email);
});
```

# LES FORMULAIRES EN JAVASCRIPT

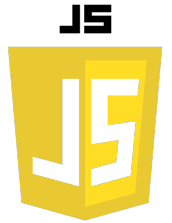
**HTML5 Validation**
HTML5 offre également des fonctionnalités intégrées pour la validation des formulaires, comme les attributs required, pattern, min, max, etc. Par exemple :

```
<form id="myForm">
  <label for="name">Nom (au moins 2 caractères):</label><br>
  <input type="text" id="name" name="name" required pattern=".{2,}"><br>
  <label for="email">Email:</label><br>
  <input type="email" id="email" name="email" required><br>
  <input type="submit" value="Submit">
</form>
```

Dans cet exemple, le champ name doit avoir au moins 2 caractères (grâce à l'attribut pattern), et le champ email doit être une adresse email valide (grâce au type email). Les deux champs sont marqués comme required, ce qui signifie que le formulaire ne peut pas être soumis si ces champs sont vides.

**Notez que la validation côté client est une bonne première étape, mais elle ne remplace pas la validation côté serveur. Un utilisateur malveillant peut facilement contourner la validation côté client, donc vous devez toujours valider les données côté serveur pour vous assurer qu'elles sont sécurisées.**

# LES FORMULAIRES EN JAVASCRIPT



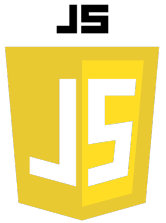

**Envoie du formulaire au serveur :**

// Crée un objet avec les données du formulaire

```
var formData = new URLSearchParams();
formData.append('name', name);
formData.append('email', email);
```

Dans ce code, nous utilisons URLSearchParams pour créer un ensemble de paires clé-valeur représentant les données du formulaire. Nous utilisons ensuite la méthode append pour ajouter chaque paire clé-valeur à l'objet formData. Enfin, nous définissons l'en-tête 'Content-Type' sur 'application/x-www-form-urlencoded' et nous passons formData comme corps de la requête.

# LES FORMULAIRES EN JAVASCRIPT




// Envoie les données du formulaire au serveur

```
fetch('https://example.com/submit', {
method: 'POST', // ou 'PUT'
headers: {
'Content-Type': 'application/x-www-form-urlencoded',
},
body: formData,
})
.then(response => response.json())
.then(data => console.log('Success:', data))
.catch((error) => console.error('Error:', error));
});
```

Cette structure logicielle est une promesse. On l'utilisera telle quelle actuellement, les promesses seront vues dans un prochain chapitre.

```
form.addEventListener('submit', function(event) {
  event.preventDefault(); // Empêche la soumission par défaut du formulaire

  var name = form.elements['name'].value;
  var email = form.elements['email'].value;

  // Crée un objet avec les données du formulaire
  var formData = new URLSearchParams();
  formData.append('name', name);
  formData.append('email', email);

  // Envoie les données du formulaire au serveur
  fetch('https://example.com/submit', {
    method: 'POST', // ou 'PUT'
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
    },
    body: formData,
  })
  .then(response => response.json())
  .then(data => console.log('Success:', data))
  .catch((error) => console.error('Error:', error));
});
```

# LES FORMULAIRES EN JAVASCRIPT

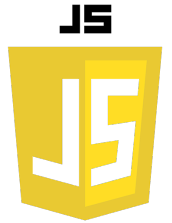
Lorsque vous envoyez des données de formulaire à un serveur, les données doivent être encodées dans un format que le serveur peut comprendre. L'un des formats les plus couramment utilisés est le format "application/x-www-form-urlencoded".

Dans ce format, les paires clé-valeur sont séparées par des caractères &, et les espaces sont remplacés par des caractères +. Par exemple, si vous avez un formulaire avec deux champs, name et email, et que l'utilisateur entre "John Doe" pour le nom et "john.doe@example.com" pour l'email, les données du formulaire encodées en URL ressembleraient à ceci :
name=John+Doe&email=john.doe%40example.com

Notez que certains caractères spéciaux sont également encodés en utilisant des codes de pourcentage. Par exemple, l'arobase (@) dans l'adresse e-mail est encodée en %40.

Lorsque vous utilisez fetch pour envoyer une requête POST avec des données de formulaire encodées en URL, vous pouvez créer une chaîne de ce type à partir d'un objet JavaScript en utilisant la classe URLSearchParams.

# RÉCUPÉRATION DES DONNÉES COTÉ SERVEUR




```
const express = require('express');
const app = express();

// Parse application/json
app.use(express.json());

app.post('/submit', (req, res) => {
    console.log(req.body);
    res.json({ message: 'Form data received' });
});

app.listen(3000, () => {
    console.log('Server is listening on port 3000');
});
```

Lorsque les données arrivent au serveur, le middleware analysera ces données encodées en URL et les convertira en un objet JavaScript, req.body pourrait ressembler à ceci :

```
{
  name: 'John Doe',
  email: 'john.doe@example.com'
}
```

Dans cet exemple, nous utilisons le middleware express.urlencoded pour analyser les corps des requêtes entrantes en données de formulaire encodées en URL. Nous définissons ensuite une route POST à '/submit' qui affiche le corps de la requête dans la console et renvoie un message de confirmation.