R102 – TP SAÉ: Formulaires HTML

En HTML, il y a des éléments dit *interactifs* car ils permettent à l'utilisateur de fournir des informations qui pourront être traitées ensuite pour afficher une nouvelle page web (ou une portion de page). Parmi ces éléments, l'un des plus complexes et puissants est sans nul doute le champ de saisie `input`, mais il en existe de nombreux autres : les zones de texte, les boutons, les listes déroulantes ou non, les boutons radios, les cases à cocher, ...

Il est préférable de regrouper ces éléments de contrôle interactif au sein d'un *formulaire* représenté par la balise `<form>`.

Le but de ce TP est donc d'introduire la notion de formulaire en HTML et les méthodes de transmission d'information du *client* vers un *serveur*.

Ordre du jour

1. Introduction

La première chose à faire est de se documenter et de tester sur des exemples simples.

Pour cela, nous vous conseillons de commencer par parcourir les différents guides du Mozilla Developer Network : https://developer.mozilla.org/fr/docs/Learn/Forms.

Pour obtenir un formulaire réussi, il faut penser à de nombreux aspects visuels, mais aussi non visuels.

Ce à quoi il faut penser :

Étiquettes

- Élément `label`
- Elles indiquent visuellement aux utilisateurs la signification des champs de saisie correspondants.
- La valeur de l'attribut optionnel `for` pourra associer cette étiquette à un contrôle qui aura son attribut `name` avec la même valeur.

Champs de saisie

- Élément `input`, `textarea`
- Les champs de saisie permettent aux utilisateurs de fournir des informations.
- Ils incluent les champs de texte, des champs de mot de passe, des cases à cocher, des boutons radio et bien plus encore.

Actions

- Élément `button` quand il est associé à un formulaire par exemple.
- Il s'agit de liens ou de boutons qui effectuent une action lorsque l'utilisateur les actionne (en cliquant dessus par exemple). Les actions classiques sont la soumission du formulaire ou sa réinitialisation.
- Ces actions sont souvent regroupées au même endroit (par exemple en fin de formulaire) afin de les trouver facilement.

Aides et messages

Ils fournissent des informations sur la façon de remplir le formulaire. On peut les trouver sous plusieurs formes qui se traduisent par divers attributs :

- Une bulle d'aide lorsque la souris fait une pause sur un élément : attribut `title` pour l'élément `input`,
- Un texte directement dans une zone de texte avant son remplissage : attribut `placeholder` ainsi que les pseudo-éléments et pseudo-classes associés `::place-holder` et `:placeholder-shown` pour sa mise en forme), un paragraphe expliquant ce qui est attendu…

  Les messages donnent des indications à l'utilisateur en fonction de sa contribution. Ils peuvent être positifs (par exemple quand il indique que le formulaire a été soumis avec succès) ou négatifs (« *Le nom d'utilisateur que vous avez sélectionné est déjà pris* »).

- Attributs `required`, `placeholder`, liste pertinente pour l'élément <*input*>,

- En css, on peut aussi afficher des informations ou modifier le visuel selon les contenues (pseudo-classes `:valid`, `:invalid`, `:checked`, pseudo-élément `::placeholder` ...). Guide à lire sur la mise en forme avec les pseudo-classes liés à l'interaction : pseudo-classe d'intéraction...

Validation intégrée

Ces mesures garantissent que les données soumises par l'utilisateur sont conformes et acceptables. Il est possible de s'assurer par exemple :

- qu'un champ est rempli (attribut `required`) avant l'envoi du formulaire
- qu'un numéro de téléphone, une adresse email ou une url est valide
- qu'un nombre ou une date soit valide et se trouve dans un intervalle accepté.

Pour plus de détails visiter cette section : Validation côté client

Validation distante

Une fois le formulaire validé côté client, les données peuvent être envoyées au serveur qui effectuera ses propres vérifications (il ne faut jamais faire confiance au client) avant de renvoyer sa réponse au client. Sa réponse peut être positive ou négative selon les cas.

Pourquoi vérifier des deux coté (coté client et coté serveur)?

1. Il ne faut JAMAIS faire confiance au client, il peut être corrompu.
2. Les validations coté client sont quasi instantanées, fournissent donc une aide rapide au remplissage et augmente les chances d'obtenir un retour positif du serveur. De cette façon, on peut éviter des transferts de données inutiles qui sont une perte de temps et de bande passante.

2. Un exemple simple

Voici un premier formulaire en guise d'exemple. **Analysez-le** en vous aidant de la documentation.

```
<form name="form1"><!-- Par défaut `action="" method="GET"`-->①
    <ul>
        <li>
            <label for="auteur">Nom : </label>②
            <input type="text" id="auteur" name="nom" maxlength="40" placeholder=
"Votre nom" title="40 caractères maximum" required>③ ④
        </li>
        <li>
            <label>⑤
                Courriel :
                <input type="email" id="email" name="email" placeholder=
"prenom.nom.etu@univ-lille.fr" pattern=".+\..+\.etu@univ-lille.fr" title="Uniquement
des emails étudiants !" required>⑥
            </label>
        </li>
        <li>
            <label for="msg">Message :</label>
            <textarea id="msg" name="user_msg" placeholder="Indiquer votre message
ici" title="Merci de rester poli"></textarea>
        </li>
        <li class="bouton">
            <input type="submit" value="Input de soumission">
            <button type="submit">Bouton de soumission</button>
            <input type="reset" value="Annuler">
            <input id="prodId" name="prodId" type="hidden" value="zx589q4312pm">⑦
        </li>
    </ul>
</form>
```

① De nombreux attributs sont importants (voir ici : Attributs de la balise *<form>*) : `action` vide = aucun envoi de données, sinon url de destination, `method` = définir comment les données seront envoyées (GET : visibles dans l'url, POST : cachées dans la requête HTTP) ...

② Étiquette rattachée à un contrôle si l'attribut `for` du label possède la valeur de l'`id` du contrôle (ici c'est l'élément `input` juste en dessous)

③ L'élément `input` est LE contrôle fourre-tout. L'attribut `type` désigne le genre de contrôle.La lecture de la documentation est le minimum : `input`

④ Noter l'attribut de validation (`maxlength`) et l'attribut `title` qui permettent d'indiquer la contrainte de validation du champ et l'affichage d'une bulle d'aide associée.

⑤ Pour les emails, la validation du format générique est automatique.

⑥ Ici, l'étiquette contient le contrôle, donc l'attribut `for` est inutile.

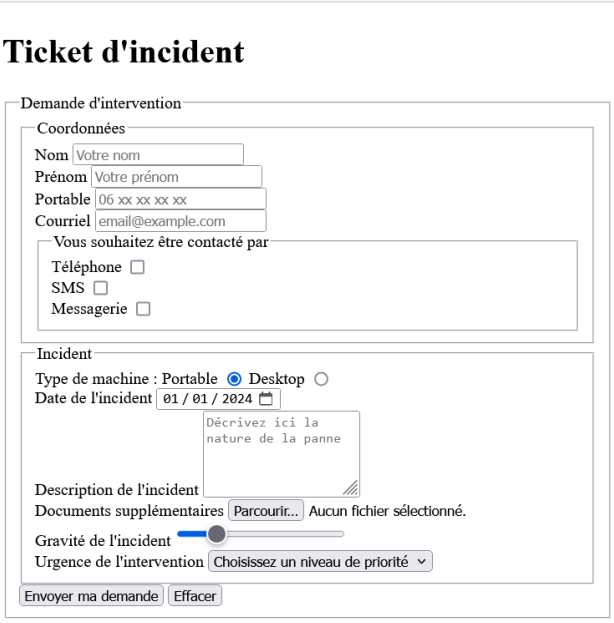
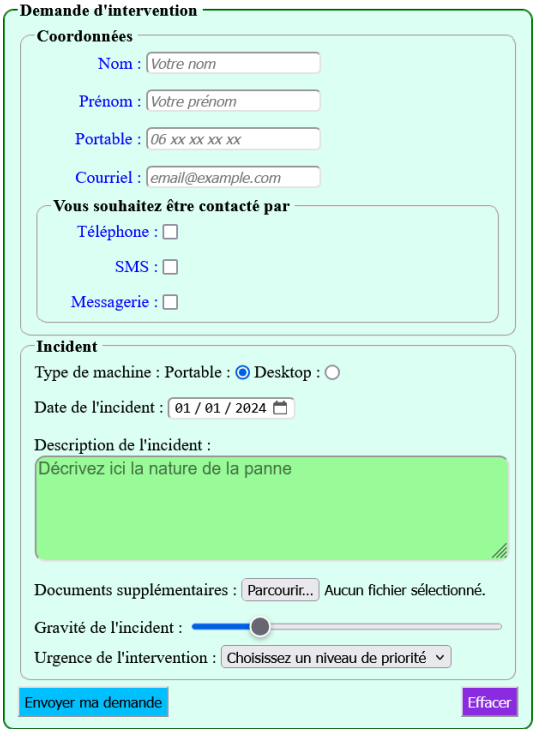
Afin de vous faire gagner du temps, voici un lien vers ce formulaire. Vous pourrez l'étudier et le modifier (HTML et CSS) : Formulaire 1

3. Un exemple plus complet

- Créer une page html (correctement) `incident.html`

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/incident.css" />
    <title>Incident</title>
</head>
<body>
    <h1>Formulaire incident</h1>
    ...
</body>
</html>
```

- Ajouter les éléments pour obtenir ce formulaire

Visuel par défaut	Visuel final
	

Vous prendrez soin de :

- Valider le champ du téléphone portable
- Mettre des bulles d'aides sur le champ de description et sur le contrôle de la *"Documentations supplémentaires"*.
- Quand un champ possède le focus, mettre le fond en rouge saumon si son contenu est invalide

- Interdire une date d'incident au-delà du 31/12/2027 (La comparaison avec la date du jour nécessite plus que HTML et CSS)
- Il faut interdire l'upload autre chose que des documents pdf ou des images.

Pour voir ce que donne vos formulaires, vous pouvez utiliser le script php joint, et ajouter `action="./afficherVariables.php"` à votre formulaire.

Pour une validation fictive, il vous suffit d'ajouter l'attribut `action` à votre balise `<form>` avec le nom d'une page HTML statique donnant l'illusion d'une validation et prise en compte de votre formulaire par le service distant.