INTRODUCTION A LA

PROGRAMMATION WEB

1ERE ANNEE DU CYCLE INGENIEUR

TD 3

Formulaire, JavaScript & Promesses

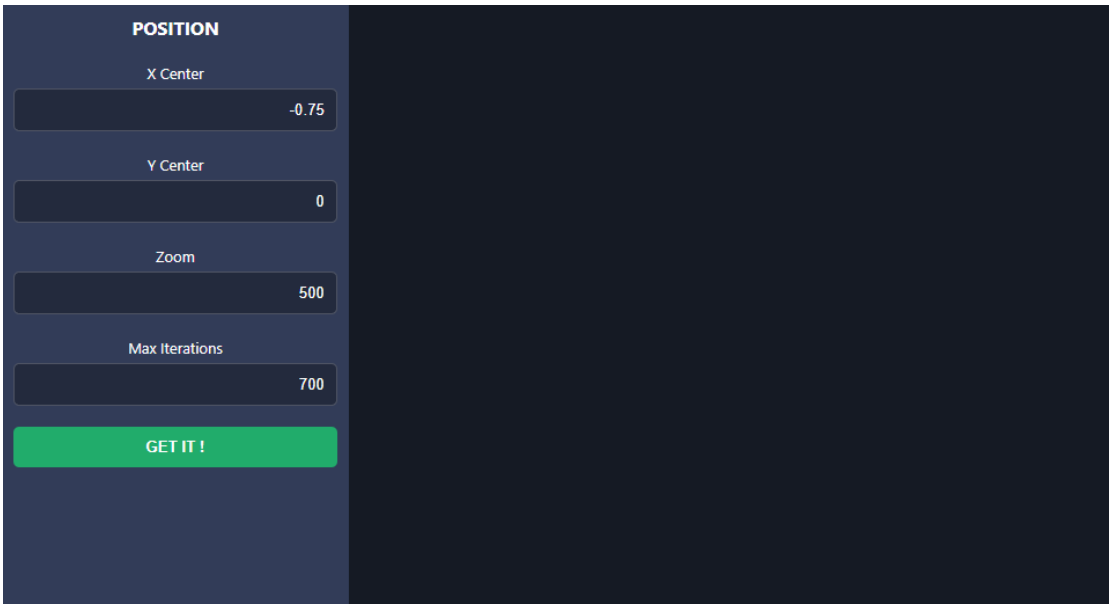
charles.meunier@u-bourgogne.fr

OBJECTIFS

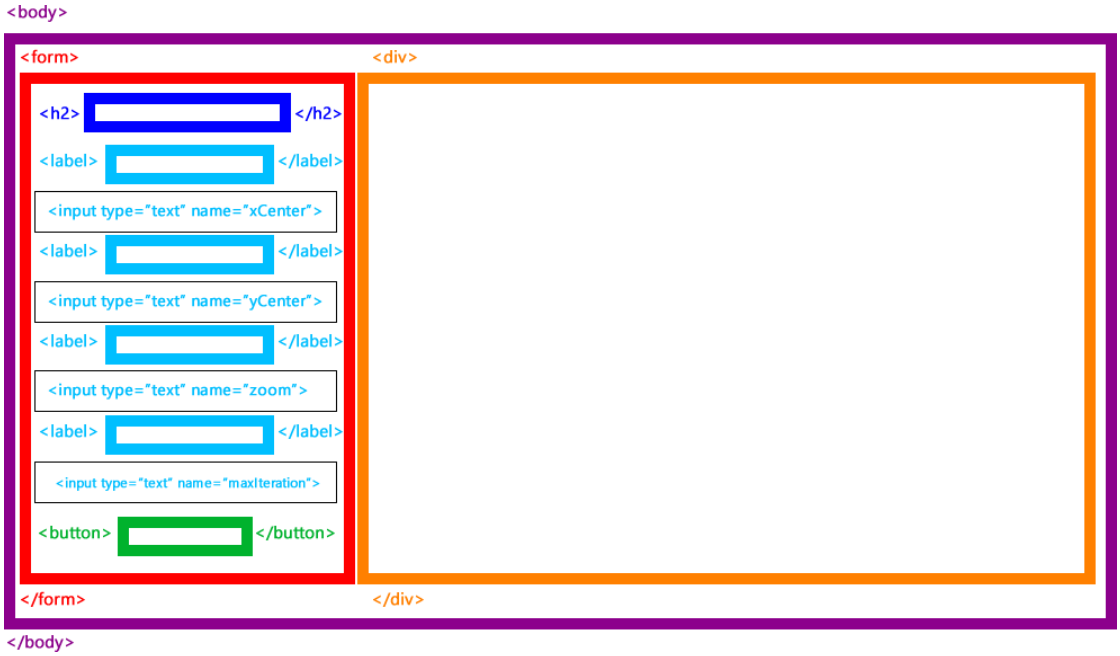
- Découvrir les formulaires et les types de champs utilisables,
- Manipuler le DOM avec JavaScript,
- Mettre en place des gestionnaires d'événements,
- Découvrir le fonctionnement des promesses.

FORMULAIRE

➢ Reproduisez l'interface ci-dessous à l'aide des informations fournies :



DECOUPAGE



TEINTES UTILISEES

- Arrière-plan de la section gauche : **#323c58**
- Arrière-plan de la section droite : **#151a24**
- Arrière-plan des champs de saisie : **#232a3d**
- Bordure des champs de saisie : **#4f5464**
- Arrière-plan du bouton : **#21ac6b**

LA BALISE <FORM>

Les formulaires sont, généralement, délimités par une balise **<form>**. Ceci indique au navigateur que tous les champs de saisie qui se trouvent au sein de la même balise **<form>** sont communs et que leurs informations doivent être transmises ensemble au destinataire.

Tout bouton ajouté au sein d'un formulaire aura pour comportement par défaut de transmettre le formulaire lorsque l'utilisateur clique dessus.

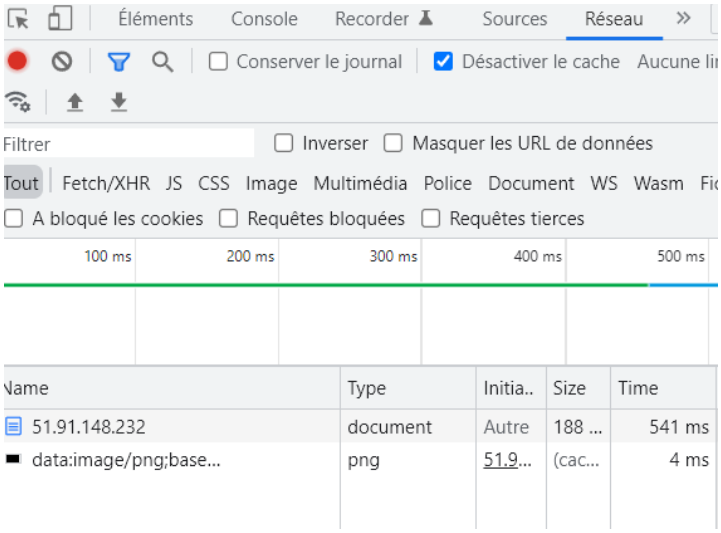
Transmettre, oui, mais OÙ ? La balise **<form>** accepte plusieurs attributs dont :

- **action** : l'URL de destination à laquelle transmettre les données du formulaire
- **method** : la méthode http à utiliser pour envoyer les données (**GET** ou **POST**)

- Demandez à l'enseignant l'adresse IP du serveur auquel transmettre les données de votre formulaire
- Ajoutez un attribut « **action="http://adresse_ip"** » à votre formulaire
- Précisez que la méthode d'envoi est **GET**
- Cliquez sur le bouton « Get it ! »

  Vous devriez être redirigé vers l'adresse IP saisie précédemment et une image devrait apparaître sur votre écran.

- Observez la barre d'adresse de votre navigateur. Son contenu correspond-il exactement à ce que vous avez saisi ?
- Modifiez la méthode d'envoi du formulaire en **POST** et procédez comme précédemment.
- Voyez-vous une différence ?
- Selon vous, quelle méthode est à privilégier pour l'envoi d'un formulaire de connexion (login / password) ?
- Ouvrez les outils de développement intégrés au navigateur (F12 ou Ctrl + Shit + i)
- Allez sur l'onglet « Réseau »
- Cliquez sur le bouton « Get it ! »
- Cliquez sur la ligne commençant par l'adresse IP communiquée précédemment :



Vous accédez aux informations échangées par le navigateur et le serveur Web distant.

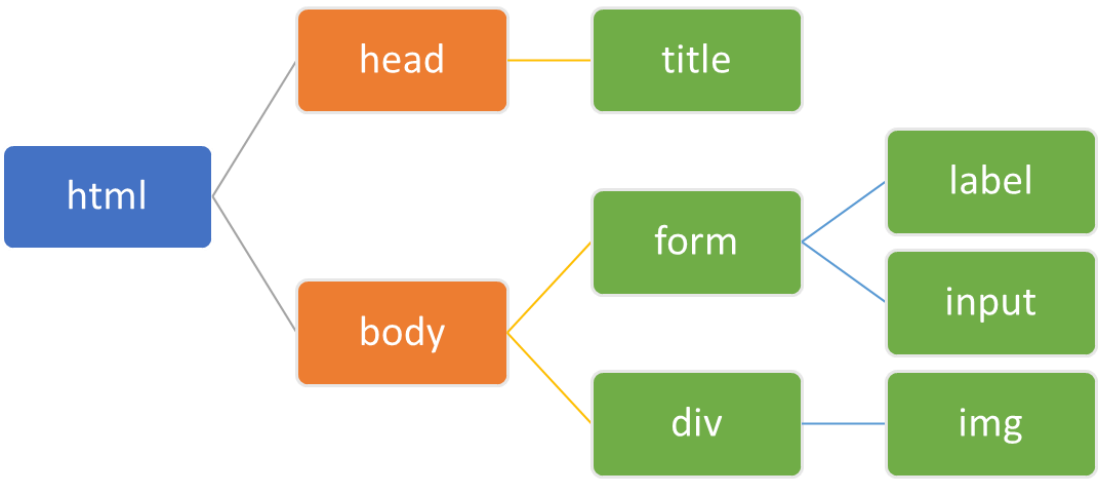
- Observez le contenu des différents onglets (entête, charge utile, …)
- Que déduisez-vous de l'envoi par la méthode **POST** ?
- Comment peut-on y remédier ?

JAVASCRIPT

JavaScript est un langage de programmation interprété par tous les navigateurs et permettant d'intégrer des traitements et de l'interaction avec l'utilisateur.

MANIPULATION DU DOM

Le DOM (Document Object Model) est la représentation de la structure de votre page Web sous la forme d'un arbre (au sens graphe du terme) dont les nœuds sont les balises HTML :



- Ajoutez un dossier « js » à la racine de votre site.
- Créez dans ce dossier un fichier « fractale.js »
- Liez votre fichier « index.html » avec le fichier précédent en ajoutant une balise **<script>** juste avant la fin du body :

```
    ...
    <script src="file-name.js"></script>
</body>
```

LA CONSOLE

La console est un outil des plus utiles pour suivre ce qu'il se passe dans votre application. Vous pouvez y afficher des informations et c'est à cet endroit que les erreurs rencontrées par votre programme apparaitront.

Pensez-donc à toujours y jeter un œil quand votre application ne se comporte pas comme vous l'attendez.

- Dans le fichier « fractale.js » ajoutez les lignes ci-dessous :

```
console.log("Fichier chargé !")
console.warn("Je te colle un avertissement !")
console.error("Oups, c'est tout cassé !")
```

- Actualisez votre page web et consultez la console des outils de développement du navigateur.

  Comme vous pouvez le voir, le script est exécuté dès son chargement.

VARIABLES

- Remplacez le contenu de votre script par le code suivant :

```
const a = 5;
const b = 10;

let c = a + b;
++a;
++c

console.log(a + " ; " + b + " ; " + c)
```

  - **let** permet de déclarer une variable
  - **const** permet de déclarer une constante

- Actualisez la page et observez la console.
- Corrigez le problème et actualisez de nouveau la console.

FONCTION

- Remplacez le contenu de votre script par le code suivant :

```
function firstFunction(parameter)
{
  console.log(`Fonction chargée (${parameter})`)
}

firstFunction(5)
```

> *Note 1 : aucun type n'est précisé quant au paramètre « **`parameter`** ». Et pour cause, JavaScript n'est pas fortement typé comme C++ par exemple. En gros, une variable peut contenir tout ce que l'on veut à un instant T et tout autre chose un peu plus tard.*

> *Note 2 : Les chaines de caractères peuvent être délimitées par des guillemets (**"**), des cotes simples (**'**) ou des anticotes (**`**). Ces dernières permettent d'intégrer des variables à l'aide de la notation **`${variable}`** sans avoir à utiliser l'opérateur de concaténation « **+** ».*

ORDRE DE CHARGEMENT

- Créez un nouveau script « script2.js » et liez-le à votre page HTML, après le lien avec « fractale.js ».
- Déplacez dans ce fichier la **déclaration** de **firstFunction**.
- Actualisez votre page et observez la console.

Deux solutions existent pour corriger ce problème :

- Lier les fichiers dans le bon ordre dans le fichier HTML de manière à ce que **`firstFunction`** soit déclarée avant d'être utilisée.
- S'assurer que tous les fichiers sont bien chargés avant de commencer les traitements.

La première solution est laborieuse et source de régressions lorsque l'on va ajouter de nouvelles fonctionnalités. Avec un ou deux fichiers c'est jouable, mais imaginez avec 500 scripts.

Avec la seconde solution, le problème ne se pose pas (sauf en cas d'héritage, mais on verra cela un peu plus tard. Parce que oui, bien que non typé, JavaScript permet de faire de la POO) car tous les fichiers sont chargés avant de commencer le traitement et donc toutes les fonctions sont connues à ce moment-là.

EVENEMENTS

- Dans le fichier fractale.js, remplacez l'appel de la fonction **`firstFunction`** par le code suivant :

```
window.addEventListener("load", () ⇒ {
  firstFunction(5)
})
```

- Actualisez la page. Le problème devrait être résolu.

> *Note 1 : **window** est un objet global JavaScript qui représente un onglet du navigateur*

> *Note 2 : **addEventListener** associe un gestionnaire d'événements à un événement produit sur l'objet cible.*

> *Note 3 : l'événement **load** est déclenché lorsque le chargement de tous les éléments de la page est terminé*

> *Note 4 : le gestionnaire d'événement est une fonction appelée lors du déclenchement de l'événement. Ici, il s'agit d'une fonction anonyme (**callback**) dont on passe directement le code en second paramètre de **addEventListener**.*

MANIPULATION DU DOM

PREPARATION

- Supprimez le fichier « script2.js » et retirer son lien dans le fichier « index.html »
- Effacez le contenu de « fractale.js »
- Dans le fichier « index.html », ajoutez l'attribut « type="button" » au bouton du formulaire.
- Actualisez la page et cliquez sur le bouton. Que se passe-t-il ?

Comme indiqué précédemment, un bouton de formulaire est de type « submit » par défaut. C'est-à-dire que lorsque l'on clique dessus, il envoie les données du formulaire à l'adresse définie dans « action » avec la méthode « method ».

A présent, vous venez de modifier le comportement par défaut du bouton qui est redevenu un simple bouton qui ne possède pas d'action associée lorsque l'on clique dessus.

ONCLICK

- Créez une méthode « onClick » qui affiche « Click ! » dans la console.
- Lorsque tous les éléments de la page seront chargés, associez la méthode « onClick » au bouton du formulaire « Get It ! » grâce au code suivant :

```
const btnGetIt = document.querySelector("button")
btnGetIt.addEventListener("click", onClick)
```

*Note 1 : **`document`** est un objet global de JavaScript représentant le DOM. Il permet d'accéder à tous les éléments constituant votre page Web*

*Note 2 : la méthode **`querySelector`** de document permet de récupérer un élément de la page à partir d'un sélecteur CSS. Ici, comme il n'y a qu'un bouton dans la page, nous pouvons directement utiliser le sélecteur CSS de la balise **`button`** pour récupérer le bouton.*

*Note 3 : on utilise ensuite **`addEventListener`** pour associer l'événement **`click`** du bouton à la méthode **`onClick`**. Notez bien l'absence de parenthèse.*

- Actualisez votre page.
- Cliquez sur le bouton « Get It ! » et observez la console.

DONNEES DU FORMULAIRE

- Dans la méthode **`onClick`**, stockez les données du formulaire dans des constantes. Par exemple :

```
const xCenter = document.querySelector("#xCenter").value
```

*Note 1 : **`document.querySelector("#xCenter")`** permet d'accéder au premier champ de saisie, à condition que ce dernier est un attribut **`id="xCenter"`**.*

*Note 2 : la propriété **`value`** retourne la valeur saisie dans le champ de formulaire.*

- Affichez dans la console le contenu des constantes précédemment créées.
- Testez

RESSOURCES ASYNCHRONES

Au début du sujet, lorsque vous transmettiez les données du formulaire, une nouvelle page était chargée dans votre onglet.

Il est possible de réaliser des requêtes vers le serveur sans procéder au rechargement complet de la page. On utilise pour cela des requêtes asynchrones parfois appelées requêtes XHR ou Ajax. Depuis ES6, JavaScript dispose de l'API fetch qui gère ce type de requêtes.

- Ajoutez le code suivant à la méthode onClick en adaptant le nom des variables si nécessaire :

```
fetch(`http://adresse-IP?xCenter=${xCenter}&yCenter=${yCenter}&zoom=${zoom}&maxIteration=${maxIteration}`)
.then((response) => {
        response.text().then((data) => {
                document.querySelector("div").innerHTML = data
        }).catch((error) =>
        {
                console.log(error)
        })
}). catch((error) =>
{
        console.log(error)
})
```

*Note 1 : le premier paramètre de **`fetch`** est l'url à laquelle on souhaite faire une requête*

*Note 2 : ici, nous utilisons la méthode **`GET`** qui nous impose de passer les paramètres dans l'url, dans ce que l'on appelle la « **`query string`** »*

*Note 3 : le **`?`** marque le début de la **`query string`** dans l'url. Le symbole **`&`** est utilisé pour délimiter les paramètres*

*Note 4 : les paramètres sont fournis avec le format **`nom=valeur`**, sans espace autour du **`=`***

La méthode fetch retourne une promesse. En effet, lorsque l'on fait une requête à un serveur, nous ne savons pas au bout de combien de temps nous allons recevoir sa réponse. Pour éviter de bloquez le fonctionnement de l'application Web, les requêtes sont traitées de manière asynchrone. Ainsi, lorsque le serveur répondra, une méthode (callback) sera appelée pour traiter sa réponse.

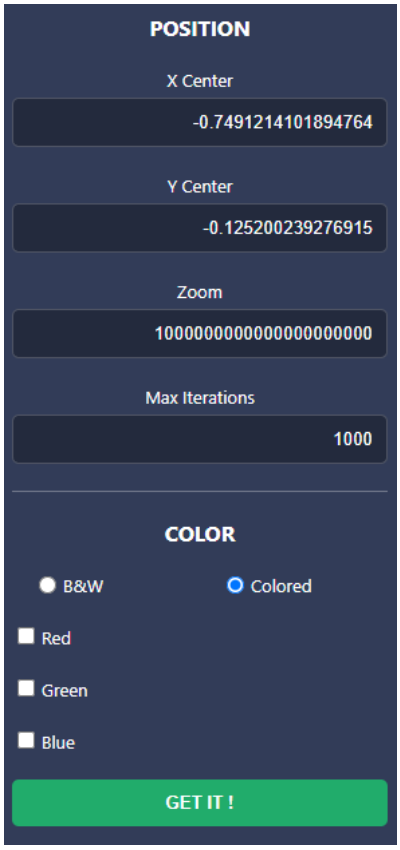
Les promesses encapsulent ce mécanisme et assurent qu'un traitement sera réalisé en cas de réussite comme en cas de problème.

```
fetch(url).then(fonction_appelee_si_ok).catch(fonction_appelee_si_erreur)
```

- Actualisez la page web et testez.

POUR ALLER PLUS LOIN

- Complétez le formulaire afin d'obtenir l'interface ci-dessous :



Note 1 : les boutons radio sont définis à l'aide de la balise ***<input type="radio">***

Note 2 : les boutons radio ayant le même attribut ***name*** *font partie du même groupe. Sélectionner l'un désélectionne les autres.*

Note 3 : les cases à cocher sont définies à l'aide de la balise ***<input type="checkbox">***

Note 4 : il est possible de savoir si une case à cocher ou un bouton radio est coché grâce à sa propriété ***checked*** *:* ***document.querySelector("#chkRed").checked***

- Faites en sorte que les composantes de couleur n'apparaissent que si l'utilisateur click sur « Colored ».

Note 1 : Il est possible d'affecter ou de retirer une classe CSS à un élément de la page avec la propriété ***classList*** *:*

```
//Ajoute la classe CSS active à #id
document.querySelector("#id").classList.add("active")
```

```
//Retire la classe CSS active à #id
document.querySelector("#id").classList.remove("active")
```

- Modifiez la méthode on`Click` pour prendre en compte les nouvelles données du formulaire :
  - **`colored`** : 1 si la bouton radio « Colored » est coché, 0 sinon.
  - **`red`** : 1 si la case à cocher « Red » est cochée, 0 sinon.
  - **`green`** : 1 si la case à cocher « Green » est cochée, 0 sinon.
  - **`blue`** : 1 si la case à cocher « Blue » est cochée, 0 sinon.