

# A propos de l'auteur

Fogan BIDI incarne l'excellence de l'ingénierie informatique en tant que chef de projet technique (CPT) au sein de Streakers Gaming Inc., une entreprise de renom basée au Canada.

En parallèle, il exerce en tant que consultant en cybersécurité, offrant son expertise à plusieurs entreprises au Togo.

De plus, Fogan est le visionnaire qui a fondé Africa Web Warriors, une organisation d'envergure abritant une prestigieuse académie. Celle-ci représente un véritable foyer pour les passionnés d'informatique, les incitant à transcender les connaissances basiques afin de mieux façonner et protéger les solutions technologiques en Afrique.

# A propos de A2W

## (Africa Web Warriors)

A2W (Africa Web Warriors) se positionne en tant que centre d'excellence dans les domaines de la recherche et du consulting spécialisés en cybersécurité et développement d'applications. Notre expertise est sollicitée par des entreprises et des particuliers à travers l'Afrique et le monde entier, afin d'assurer la protection de leurs réseaux informatiques et applications.

Dans cette optique, nous avons créé une prestigieuse académie au sein d'A2W, offrant une opportunité unique à tous ceux souhaitant s'immerger dans le monde de la cybersécurité (hacking éthique) ou du développement d'applications, qu'elles soient web, mobiles ou desktop.

Notre académie propose des certificats de renom :

## **JEH (Junior Ethical Hacker)**

Explorez les arcanes du hacking éthique en devenant un hacker éthique junior. Vous maîtriserez les méthodes utilisées par les experts pour identifier les failles de sécurité et protéger les systèmes informatiques des entreprises. Grâce à votre expertise, vous serez en mesure d'évaluer la vulnérabilité des systèmes existants et de proposer des solutions pour renforcer la sécurité des données.\*\*\*\*

Durée : 6 mois

---

## **JWD (Junior Web Developer)**

Ce certificat vous ouvre les portes de la création d'applications en vous dotant de solides bases. À la fin de ce programme, vous serez en mesure de concevoir des projets web de taille moyenne. Il constitue également une étape préalable pour une transition harmonieuse vers le développement d'applications mobiles.

Durée : 3 mois

---

## **CWD (Certified Web Developer)**

Ce certificat vous ouvrira les portes de la création d'applications en vous dotant de solides bases. À la fin de ce programme, vous serez capable de concevoir des projets web de taille

moyenne. Il constitue également une étape préalable pour une transition harmonieuse vers le développement d'applications mobiles.

Durée : 6 mois

---

## **CMD (Certified Mobile Developer)**

Obtenez la certification de développeur web certifié et positionnez-vous en tant qu'expert chevronné sur le marché du développement d'applications web. Vous serez apte à mener et superviser des projets d'envergure avec assurance et professionnalisme.

Durée : 6 mois

Nous espérons que ces programmes répondront à vos aspirations !

---

## **Nous contacter**

Adresse : Lomé - Togo

Tel : +228 91 56 75 90

Mail : [contact@africa-web-warriors.org](mailto:contact@africa-web-warriors.org)

Site web : [www.africa-web-warriors.org](http://www.africa-web-warriors.org)

# Introduction

Découvrez le monde fascinant des plateformes web qui façonnent notre quotidien ! Chaque jour, nous nous appuyons sur ces outils polyvalents pour satisfaire nos besoins divers, allant de la simple messagerie aux appels vidéo captivants (Google, Gmail, Facebook, YouTube, et bien d'autres).

Mais avez-vous déjà songé à ce qui se cache derrière ces merveilles technologiques ? Dans les coulisses de ces plateformes, des développeurs et programmeurs talentueux ont consacré leurs compétences pour créer des univers interactifs qui améliorent notre expérience en ligne.

Si vous tenez ce livre entre vos mains, c'est que vous aspirez certainement à acquérir ces compétences vous permettant de matérialiser vos idées et de concevoir des solutions novatrices, que ce soit par passion ou par ambition.

Vous êtes au bon endroit ! Au fil des chapitres qui suivent, nous vous guiderons avec clarté et concision pour vous enseigner tout ce qu'il faut savoir afin de concevoir aisément une plateforme ou une application web à la hauteur de vos ambitions.

Laissez-vous emporter par l'architecture captivante du web. Avant de vous lancer dans la création d'une application ou d'une plateforme qui y sera hébergée, plongez dans une compréhension globale de son fonctionnement. Explorez les arcanes du développement, de l'interaction entre le client et le serveur, et découvrez comment ces éléments s'assemblent harmonieusement pour offrir une expérience en ligne captivante.

En vous initiant au **Front-end**, vous maîtriserez l'art de concevoir l'interface utilisateur, tandis que le **Back-end** vous dévoilera les secrets d'un monde dynamique, faisant le lien entre le client et le serveur. Avec ces compétences en poche, vous serez prêt à créer des plateformes interactives et à faire naître des contenus captivants pour vos utilisateurs.

Plongez dans l'univers passionnant des plateformes web et transformez vos idées en réalité grâce à ce guide inestimable.

# HTML

(Hyper Text Markup Langage)

# Introduction

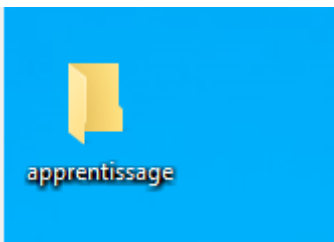
HTML est l'un des langages informatiques interprétés par les navigateurs. Il nous permet de construire le contenu visible dans le navigateur.

Lorsque nous visitons des sites web tels que youtube.com, nous pouvons voir du texte, des images, des vidéos, des zones de saisie, etc. Tous ces éléments sont créés à l'aide de HTML. Nous allons maintenant découvrir comment accomplir cela.

## Organisez votre projet

Pour chaque projet web que vous créez, il est important de bien organiser vos fichiers sur votre ordinateur. Vous devez créer un dossier dédié à chaque projet que vous souhaitez développer.

Dans le cadre de ce cours, nous vous demandons de créer un dossier nommé "apprentissage" sur votre bureau ou dans tout autre emplacement de votre choix sur votre ordinateur.



## Outils à installer sur votre ordinateur

Avant de continuer, vous devez avoir obligatoirement les deux outils suivants sur votre ordinateur:

- Un éditeur de code
- Un navigateur

Tout ordinateur vient par défaut avec au moins un navigateur (par exemple internet explorer sur les systèmes Windows).

Mais vous pouvez télécharger les navigateurs comme Firefox, Google Chrome, Opéra ou tout autre navigateur de votre choix.

En ce qui concerne l'éditeur de texte, là encore vous avez plusieurs choix:

- Notepad++
- Atom
- Sublime Texte
- Visual studio code

- etc...

Si vous êtes débutant et que vous n'avez aucune idée de quel éditeur choisir, vous pouvez opter pour "Visual studio code". Il vous aidera à aller beaucoup plus rapidement dans la rédaction de vos codes.

## Architecture minimale d'une page web

Avant de poursuivre, assurez-vous d'avoir un éditeur de texte tel que Notepad++ ou Visual Studio Code installé sur votre ordinateur.

Chaque projet web doit comporter un fichier "index.html" en tant que premier fichier. Nous allons donc créer ce fichier dans notre dossier "apprentissage" et y ajouter le contenu suivant dans l'éditeur de texte que vous avez installé sur votre ordinateur :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mon projet d'apprentissage web</title>
    <meta charset="utf-8" />
  </head>
  <body>
    Bienvenue sur ma page web!!!
  </body>
</html>
```

Si vous ouvrez ce fichier dans un navigateur en double-cliquant dessus, vous verrez son contenu s'afficher.

Avant d'ouvrir le fichier dans le navigateur, analysons l'architecture minimale de la page web que nous avons définie.

La première ligne `<!DOCTYPE html>` indique au navigateur que nous utilisons la version 5 de HTML. En effet, il existe différentes versions de HTML.

Les éléments `<html></html>` englobent tout le contenu de la page web. Elles marquent respectivement le début et la fin de notre document ou page web, qui est "index.html" dans notre cas.

Une page web est composée d'une section `head` et d'une section `body`. Dans notre architecture minimale, vous pouvez les identifier grâce aux éléments `<head></head>` et `<body></body>`.

La section "head" contient des informations sur la page web, telles que le titre ( `<title>` ) et les méta-informations ( `<meta>` ). Ces méta-informations peuvent régler certains problèmes



d'encodage.

La section "body" contient le contenu visible dans le navigateur.

Toutes les pages web que vous créerez suivront cette architecture minimale. Il est important de comprendre et de savoir écrire ce code, même avec les yeux fermés.

## Notion de balise

Dans notre code "index.html", vous avez remarqué l'utilisation de éléments tels que `<html>`, `</html>`, `<head></head>`, `<body></body>`, etc. Ces éléments sont appelés des balises.

HTML utilise des balises pour construire les éléments visibles dans une page web. Chaque élément visible est créé à partir d'une balise ou d'une combinaison de plusieurs balises.

Par exemple, si vous souhaitez afficher une image sur votre page web, vous devez rechercher la balise HTML spécifique pour l'affichage des images.

## Les différents types de balises

Dans notre fichier "index.html", vous avez pu observer deux types de balises.

La balise `<meta>` est écrite différemment des autres balises :

```
<meta />
```

Toutes les autres balises ont une balise d'ouverture et une balise de fermeture marquée par un slash (/) :

```
<html></html>
<head></head>
<title></title>
<body></body>
```

La différence de notation indique que certaines balises sont de type "paire" (avec une balise d'ouverture et une balise de fermeture), tandis que d'autres sont de type "singleton" (comme `<meta />`).

## Notion d'attributs et de valeurs

Prenons l'exemple de la balise `<meta>` pour illustrer la notion d'attribut et de valeur. Il est important de comprendre cette notion, car elle représente l'un des fondements d'HTML.

```
<meta charset='utf-8' />
```

Certaines balises, qu'elles soient de type paire ou singleton, nécessitent obligatoirement un ou plusieurs attributs pour fonctionner. Par exemple, la balise `<meta>` utilisée seule, sans attributs, n'a aucune action spécifique :

```
<meta />
```

Il est donc nécessaire de spécifier à la balise `<meta>` ce qu'elle doit faire ou accomplir sur la page web. Dans l'exemple ci-dessus :

- `charset` est l'attribut,
- `utf-8` est la valeur.

Il est important d'encadrer les valeurs entre guillemets ( `" "` ) ou apostrophes ( `' '` ).

## Comment identifier la balise à utiliser pour créer un composant ?

C'est simple. Il vous suffit de faire une petite recherche sur Google pour trouver la balise appropriée ainsi que les attributs et les valeurs associées. Chaque fois que vous vous trouvez dans cette situation, il est important de développer le réflexe de chercher par vous-même. C'est ainsi que vous apprendrez et progresserez dans ce domaine.

Ce livre n'a pas pour but de vous encombrer avec des centaines de balises ou de concepts que vous n'utiliserez peut-être pas dans vos projets. Notre objectif est de vous fournir une compréhension solide du langage et du développement web, ainsi que de vous encourager à mener vos propres recherches. La capacité à effectuer des recherches autonomes est essentielle pour tout développeur.

Une fois que vous maîtriserez les principes fondamentaux du langage, vous pourrez apprendre d'autres concepts avancés si votre projet l'exige.

Vous apprendrez davantage en réalisant des projets personnels, il est donc inutile de vous bombarder avec une multitude de balises. Pas besoin de les mémoriser, vous les intégrerez naturellement au fil du temps en réalisant de petits projets personnels.

## Exercices pratiques

Pour renforcer votre apprentissage d'HTML, voici une série de codes illustrant l'utilisation de différentes balises. Essayez de saisir ces codes dans la section "body" de votre fichier

"index.html" pour voir les résultats et comprendre le rôle des balises utilisées. N'hésitez pas à faire des recherches sur les balises qui vous semblent complexes ou confuses.

## Un lien pour accéder à Google

```
<a href="https://google.com">Cliquez ici pour accéder à Google</a>
```

## La liste des langages front-end

```
<ol>
  <li>HTML</li>
  <li>CSS</li>
  <li>Javascript</li>
</ol>
```

## Le logo de A2W

```

```

## Un formulaire de connexion

```
<form>
  <input type="email" placeholder="Votre email" />
  <input type="password" placeholder="Votre mot de passe" />
  <br />
  <input type="submit" value="Se connecter" />
  <input type="reset" value="Annuler" />
</form>
```

## La liste des langages front-end sous forme de tableau

```
<table border="1">
  <thead>
    <tr>
      <th>Rang</th>
      <th>Nom</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>1</td>
      <td>HTML</td>
```

```

</tr>
<tr>
  <td>2</td>
  <td>CSS</td>
</tr>
<tr>
  <td>3</td>
  <td>Javascript</td>
</tr>
</tbody>
</table>

```

## Quel est votre langage préféré ?

```

<input type="radio" name="choix" />HTML <br />
<input type="radio" name="choix" />CSS <br />
<input type="radio" name="choix" />Javascript

```

## Quels sont vos langages préférés ?

```

<input type="checkbox" />HTML <br />
<input type="checkbox" />CSS <br />
<input type="checkbox" />Javascript

```

# CSS

(Cascading Style Sheets)

# Introduction au CSS

CSS est un langage essentiel qui va de pair avec HTML. Il est presque impossible de créer une page web professionnelle sans utiliser CSS.

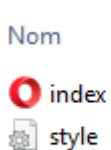
CSS ajoute du style à nos composants HTML, nous permettant d'améliorer leur apparence en ajoutant des couleurs, des formats, des polices, et bien plus encore. Il nous permet de rendre nos éléments visuellement attrayants et attractifs par rapport à leur apparence par défaut. CSS nous permet également d'ajouter des animations à nos éléments HTML.

En résumé, CSS est un allié indispensable dans votre parcours de développement web.

## Intégration du CSS dans notre page

Tout comme HTML, nous devons créer un fichier séparé pour écrire notre code CSS.

Pour cela, créons un fichier nommé `style.css` dans notre dossier d'apprentissage.



Si vous double-cliquez sur ce fichier `style.css` pour l'ouvrir, il s'ouvrira dans un éditeur de code (probablement le bloc-notes). Cela signifie que nous ne pouvons pas exécuter directement le code CSS comme nous l'avons fait pour le fichier `index.html`.

Dans ce cas, vous vous demandez peut-être :

***Comment exécuter le code CSS que nous allons écrire dans le fichier `style.css` ?***

Comme nous l'avons mentionné précédemment, CSS fonctionne de concert avec HTML. Étant donné que nous pouvons exécuter le contenu du fichier HTML ( `index.html` ) dans un navigateur, nous devons inclure le fichier CSS ( `style.css` ) dans le fichier HTML pour que le code CSS s'exécute en même temps que la page HTML dans le navigateur.

Voici comment nous pouvons inclure le fichier `style.css` dans la page `index.html` :

```
<link rel="stylesheet" href="style.css" />
```

La balise `<link>` est utilisée en combinaison avec deux attributs ( `rel` et `href` ).

Voici le code complet de `index.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mon projet d'apprentissage web</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    Bienvenue sur ma page web!!!
  </body>
</html>
```

Comme pour le moment rien n'est écrit dans le fichier `style.css`, il n'y aura pas de changement visible dans le navigateur.

## À quoi ressemble le code CSS ?

Avant de répondre à cette question, essayons de comprendre comment CSS fait son travail dans une page web.

CSS cible un élément HTML (par exemple, `body`) et définit un attribut (par exemple, la couleur de fond) avec une valeur spécifique (par exemple, `bleu`).

La syntaxe de CSS se résume dans cet exemple explication que nous venons de donner :

```
body {
  couleur-de-fond: bleu;
}
```

Cependant, nous convenons que nous n'allons pas utiliser `couleur-de-fond` comme nom d'attribut et `bleu` comme valeur. L'exemple ci-dessus est à titre d'illustration, mais l'idée qui se cache derrière CSS y est présente.

Voici ce que nous devons réellement écrire :

```
body {
  background-color: blue;
}
```

La bonne nouvelle est que toutes les balises peuvent être utilisées à la place de `body` dans l'écriture ci-dessus.

Dans ce code, `body` est appelé le `sélecteur` d'un point de vue technique.

# En savoir plus sur les sélecteurs CSS

Le sélecteur est un élément clé du bon fonctionnement d'un code CSS. Il représente l'élément ou les éléments HTML qui seront modifiés ou arrangé par CSS. Il est donc important de choisir les sélecteurs avec soin.

Nous savons déjà que nous pouvons utiliser le nom de balise comme sélecteur, par exemple :

```
/* Définir la taille des <input /> à 12 pixels */
input {
  font-size: 12px;
}

/* Définir la largeur des <img /> à 120 pixels et la hauteur à 100 pixels */
img {
  width: 120px;
  height: 100px;
}
```

Dans cet exemple, nous avons utilisé les balises `input` et `img` comme sélecteurs. Cependant, il existe deux autres types de sélecteurs que vous devez absolument connaître. Il s'agit des sélecteurs basés sur l'identifiant ( `id` ) et la classe ( `class` ) des éléments HTML.

Considérons le code suivant :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mon projet d'apprentissage web</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div id="zone">Je suis un bloc de texte</div>
    <p class="note">Je suis un paragraphe</p>
  </body>
</html>
```

Nous utilisons ici respectivement les attributs `id` et `class` sur les balises `div` et `p`. Comme vous pouvez le deviner, `id` et `class` sont des attributs.

Faites une petite recherche rapide sur Google pour en savoir plus sur ces deux attributs ( `id` et `class` ).



Maintenant, essayons de changer la couleur du texte ayant pour `id` "zone" en rouge et la couleur du texte ayant la `class` "note" en vert :

```
#zone {  
  color: red;  
}  
  
.note {  
  color: green;  
}
```

Si vous enregistrez le tout ( `Ctrl + s` ) et rafraîchissez la page dans le navigateur, vous devriez voir le résultat suivant :

Je suis un bloc de texte

Je suis un paragraphe

## Où trouver d'autres attributs CSS ?

Voici une liste non exhaustive de certains attributs CSS :

- color
- font-size
- display
- border
- text-align
- font-weight
- width
- height
- overflow
- margin
- padding
- background.

Effectuez une recherche sur Google pour savoir comment utiliser ces différents attributs que nous avons énumérés et pour découvrir d'autres attributs.

Néanmoins, n'essayez pas de tous les mémoriser. Vous les intégrerez au fur et à mesure que vous progressez dans le développement web.

# Aller plus loin avec CSS

Après avoir maîtrisé les bases de CSS, il est temps de passer à la vitesse supérieure en utilisant un Framework CSS.

Un Framework CSS vous permet d'écrire du code CSS rapidement, ce qui améliore votre productivité.

Parmi ces Framework, nous avons `Bootstrap` et `Tailwind CSS`, qui font partie des plus populaires.

Nous vous recommandons d'utiliser `Tailwind CSS`, car il est léger et très facile à prendre en main.

## Introduction à Tailwind CSS

### Qu'est-ce que Tailwind CSS ?

Tailwind CSS est un outil qui vous aide à styler vos sites web plus facilement. C'est un peu comme une boîte à outils qui contient différentes classes prédéfinies pour ajouter des styles à vos éléments HTML. Au lieu d'écrire beaucoup de code CSS personnalisé, vous pouvez simplement utiliser ces classes pour styler vos éléments rapidement.

### Pourquoi utiliser Tailwind CSS ?

Tailwind CSS a plusieurs avantages :

1. **Facilité d'utilisation** : Avec Tailwind CSS, vous n'avez pas besoin de créer beaucoup de CSS vous-même. Vous pouvez simplement ajouter des classes à vos éléments pour leur donner des styles.
2. **Personnalisation** : Tailwind CSS est également très flexible. Vous pouvez modifier les styles par défaut ou même créer vos propres classes personnalisées si vous le souhaitez.
3. **Construction rapide** : Grâce aux classes prédéfinies de Tailwind CSS, vous pouvez construire des sites web plus rapidement. Vous n'avez pas besoin d'écrire autant de code CSS à partir de zéro.

## Utilisation de Tailwind CSS

Pour commencer à utiliser Tailwind CSS, vous avez deux options : utiliser des liens spéciaux dans votre fichier HTML ou installer Tailwind CSS avec `npm`.

Dans ce cours, nous allons nous concentrer sur la première option, qui est plus facile pour les débutants.

### Utilisation des liens spéciaux

1. Dans votre fichier HTML, ajoutez les liens spéciaux pour Tailwind CSS dans la section

`<head>` :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mon projet d'apprentissage web</title>
    <meta charset="utf-8" />
    <link href="https://cdn.tailwindcss.com/2.2.19/tailwind.min.css"
rel="stylesheet">
  </head>
  <body>
    <!-- Le reste du contenu de votre page -->
  </body>
</html>
```

2. Maintenant, vous pouvez utiliser les classes spéciales de Tailwind CSS dans votre code HTML en les ajoutant directement à vos éléments. Par exemple, pour changer la couleur d'un paragraphe en rouge, vous pouvez ajouter la classe `text-red-500` :

```
<p class="text-red-500">Je suis un paragraphe rouge</p>
```

## Utilisation des classes spéciales

Tailwind CSS propose de nombreuses classes spéciales pour styliser vos éléments HTML.

Voici quelques exemples de classes couramment utilisées :

- `text-{color}` : Change la couleur du texte, par exemple `text-red-500`.
- `bg-{color}` : Change la couleur de fond, par exemple `bg-gray-200`.
- `font-{size}` : Change la taille de la police, par exemple `font-sm`.
- `p-{size}` : Ajoute un espacement intérieur (padding) à un élément, par exemple `p-4`.
- `m-{size}` : Ajoute une marge (margin) à un élément, par exemple `m-2`.
- `w-{size}` : Change la largeur d'un élément, par exemple `w-full`.
- `h-{size}` : Change la hauteur d'un élément, par exemple `h-64`.
- `flex` : Utilise le système de flexbox pour un élément.
- `items-{alignment}` : Aligne les éléments fils, par exemple `items-center`.

Vous pouvez combiner ces classes pour obtenir les styles que vous souhaitez. Par exemple, pour créer une boîte avec un fond rouge, du texte blanc et une marge de 4 unités, vous pouvez utiliser les classes suivantes :

```
<div class="bg-red-500 text-white m-4">Contenu de la boîte</div>
```

## Ressources supplémentaires

Pour en savoir plus sur Tailwind CSS et découvrir toutes ses fonctionnalités, vous pouvez consulter les ressources suivantes :

- Site officiel de Tailwind CSS : <https://tailwindcss.com/>
- Documentation de Tailwind CSS : <https://tailwindcss.com/docs>
- Tailwind Play : <https://play.tailwindcss.com/>

Ces ressources vous aideront à explorer davantage Tailwind CSS et à maîtriser l'utilisation de ses classes spéciales pour créer des sites web attrayants et réactifs.

N'oubliez pas de pratiquer et d'expérimenter Tailwind CSS pour vous familiariser davantage avec ses fonctionnalités et tirer le meilleur parti de cet outil puissant.

# Javascript

# Introduction

L'un des langages qui est de plus en plus utilisé dans le développement des applications web (mais aussi mobile et desktop) est bien évidemment JavaScript.

En tant qu'un développeur web qui se respecte, vous vous devez d'avoir une maîtrise au moins moyenne de ce langage JavaScript.

Il apporte beaucoup d'interactivité entre votre application et les utilisateurs. Vous pouvez aussi effectuer certains contrôles des données utilisateurs avec ce langage.

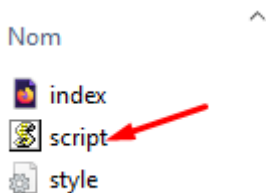
Mieux encore, les Framework modernes qui sortent de nos jours sont majoritairement basés sur JavaScript.

Bref, investir son temps à apprendre JavaScript vous sera très bénéfique dans vos futurs projets web.

## Intégration de JavaScript dans HTML

Avant de continuer, nous allons adopter la même procédure vue précédemment dans le chapitre concernant CSS.

Créons un fichier nommé `script.js` dans notre dossier apprentissage



Ce fichier `script.js`, comme le fichier `style.css`, ne peut pas s'exécuter en double-cliquant dessus.

Il faudra trouver un moyen pour l'intégrer dans la page `index.html` comme nous l'avons fait pour le fichier `style.css`.

Voici ci-dessous le code qui nous permet d'inclure ce fichier JavaScript dans la page web (`index.html`) qui lui peut-être visible dans le navigateur :

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

À ce stade, voici à quoi ressemble le fichier `index.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mon projet d'apprentissage web</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>
  <body>
    Bienvenue sur ma page web!!!
  </body>
</html>
```

Bien évidemment, comme nous n'avons rien écrit dans ce fichier `script.js`, il n'y aura rien qui va se passer si nous actualisons le contenu de `index.html` dans le navigateur.

`À titre d'exemple, ouvrons le fichier `script.js`` et mettons-y le code suivant :

```
alert("Bonjour tout le monde");
```

Si nous actualisons la page une fois encore, on peut voir une fenêtre modale qui s'affiche avec pour message `Bonjour tout le monde`.



Dans ce code, `alert` représente une `fonction` qui est intégrée dans le langage JavaScript que nous pouvons utiliser pour afficher des informations.

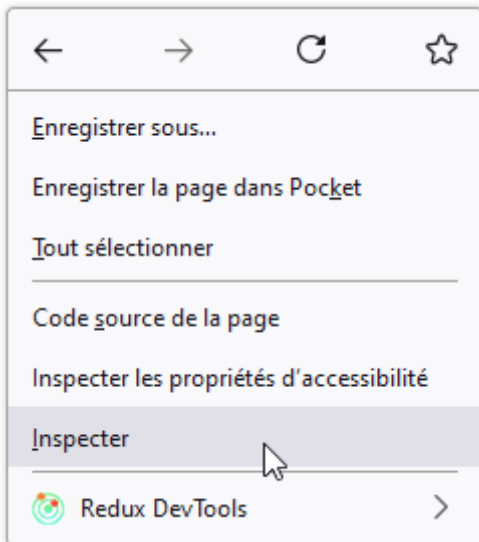
L'une des fonctions native que vous aurez à utiliser souvent en tant que développeur web est `console.log()`.

Elle vous permettra d'afficher des informations comme la fonction `alert` mais pas dans une fenêtre modale.

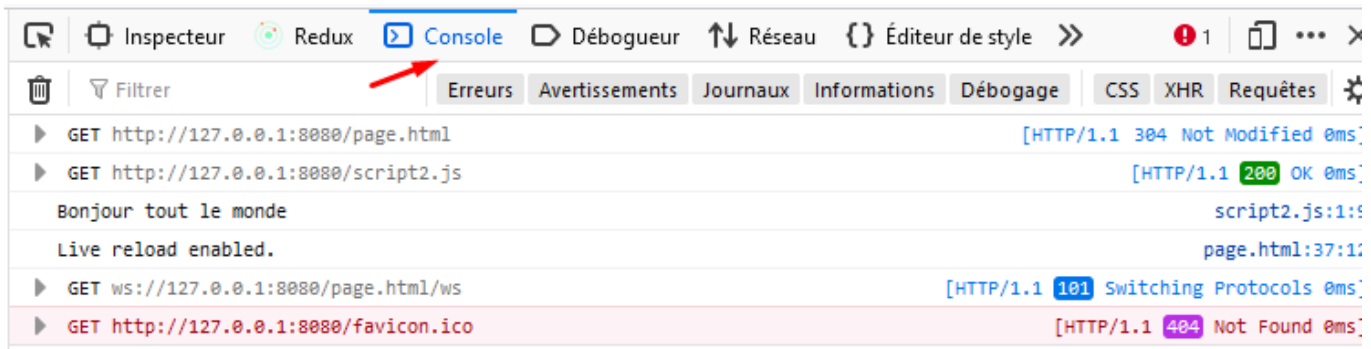
```
console.log("Bonjour tout le monde");
```

La fonction `console.log` affiche la chaîne de caractères qui lui est passée en paramètre dans la partie `console` de votre navigateur.

Faites un clic droit dans le navigateur et choisir inspecter



Ensuite, dans l'onglet qui va apparaître, cliquez sur `console`



Nous utilisons souvent cette fonction pour traquer des erreurs éventuelles lors du développement des applications.

## Le paradigme de programmation fonctionnelle.

Pour réussir à écrire du bon code JavaScript, nous vous conseillons d'adopter le paradigme de la programmation fonctionnelle.

Ce paradigme stipule tout simplement qu'il faut lier une action à une fonction.

Exemple : Supposons que sur votre page web, vous demandez à l'utilisateur de saisir un nombre `A` et un nombre `B`, et ensuite cliquer sur le bouton `Calculer la somme`.



Nombre A

Nombre B

Calculer la somme

Selon le paradigme de la programmation fonctionnelle, lorsque l'utilisateur va cliquer sur le bouton, vous devez faire appel à une fonction qui va effectuer le calcul et afficher le résultat.

Nombre A

Nombre B

Calculer la somme

La somme est 20

## Scenario

Étant donné l'importance de ce langage dans le développement web, nous n'allons pas l'apprendre comme nous avons appris le HTML et le CSS.

Nous allons vraiment réaliser un projet et apprendre de façon pratique comment utiliser JavaScript dans un vrai projet.

Voici ci-dessous le projet que nous allons construire :

Interface initiale

## Table de multiplication

Saisir un nombre

Afficher

Annuler

L'utilisateur saisie le nombre

# Table de multiplication

Saisir un nombre

Afficher

Annuler

L'utilisateur clique sur le bouton "Afficher"

## Table de multiplication

Saisir un nombre

Afficher

Annuler

$6 \times 0 = 0$

$6 \times 1 = 6$

$6 \times 2 = 12$

$6 \times 3 = 18$

$6 \times 4 = 24$

$6 \times 5 = 30$

$6 \times 6 = 36$

$6 \times 7 = 42$

$6 \times 8 = 48$

$6 \times 9 = 54$

$6 \times 10 = 60$

$6 \times 11 = 66$


Comme le schéma ci-dessus le montre, nous avons une petite application qui demande à l'utilisateur de saisir un nombre, et ensuite afficher la table de multiplication du nombre saisi après avoir cliqué sur le bouton `Afficher`.

Le bouton `Annuler` permet de réinitialiser l'interface de notre application, une fonction sera aussi créée pour jouer ce rôle d'annulation.

Sans trop tarder, mettons-nous au travail.

## Mise en place de notre projet

Nous allons créer un nouveau dossier pour ce projet. Je vais nommer ce dossier `multiplication`. Je vous conseille de faire pareil sur votre ordinateur.

 `multiplication`

Dans ce dossier, créons les trois fichiers respectifs depuis notre éditeur de code :

- index.html
- style.css
- script.js



Ouvrons le fichier `index.html` et mettons-y le code de l'architecture minimale en incluant en même temps les fichiers `style.css` et `script.js` :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Projet table de multiplication</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>
  <body>

  </body>
</html>
```

Nou allons d'abord construire l'interface utilisateur que voici :

## Table de multiplication

Saisir un nombre

Afficher

Annuler

Essayez de reproduire cette interface seul avant de regarder ce que nous allons proposer. Ce n'est pas grave si vous ne réussissez pas. Vous êtes en train d'apprendre et c'est cela le plus important.

Voici le code de l'interface utilisateur contenu dans le code `index.html` :

```

<!DOCTYPE html>
<html>
  <head>
    <title>Projet table de multiplication</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>
  <body>
    <div class="form">
      <h1>Table de multiplication</h1>

      <p>Saisir un nombre</p>
      <input type="text" id="nombre" />

      <br /><br />

      <button>Afficher</button>
      <button>Annuler</button>

      <div id="resultat"></div>
    </div>
  </body>
</html>

```

Comme vous pouvez le voir, on a essayé de rester très simple, nous avons écrit quelques codes CSS qui essayent de mettre en forme certains composants HTML.

Voici le contenu du fichier `style.css` :

```

body{
  display:flex;
  justify-content:center;
  align-items:center;
}

p{
  margin-bottom:0px;
  font-weight:bold;
}

input{
  padding:5px;
  border-radius:5px;
  border:1px solid silver;
}

```

```

outline:none;
}

button{
padding:6px;
border:none;
border-radius:5px;
}

#resultat{
margin-top:10px;
}

```

Regardez ces deux fichiers et essayez de bien comprendre ce qui est fait.

Pour le contenu CSS, si certains attributs vous paraissent flous, faites une recherche Google pour mieux les comprendre et savoir quel rôle ils jouent dans notre application.

Jusqu'alors, le fichier `script.js` est vide, nous n'avons encore rien écrit dans ce fichier.

Occupons-nous maintenant de la partie dynamique de notre application.

L'utilisateur doit saisir un nombre dans la zone de saisie. Quand il va cliquer sur le bouton `Afficher` on doit faire l'affichage de la table de multiplication du nombre saisi.

Il faudra trouver un moyen pour indiquer à notre programme qu'une fonction JavaScript va s'occuper du calcul quand on va cliquer sur le bouton.

Nous devons dans ce cas utiliser l'attribut `onclick` sur le bouton `Afficher`. Cet attribut aura pour valeur une fonction JavaScript que nous devons créer dans le fichier `script.js`.

Voici comment nous pouvons le faire:

```

<button onclick="afficher_resultat()">Afficher</button>

```

Comme nous pouvons le voir, l'attribut `onclick` a pour valeur `afficher_resultat( )` qui est une fonction que nous devons créer.

Nous savons que c'est une fonction à cause de la parenthèse qui suit le nom `afficher_resultat`.

Allons dans le fichier `script.js` et créons cette fonction :

```

const afficher_resultat={()=>{
  alert("Hello");
}

```

```
}
```

Nous avons mis une alerte dans la fonction pour tester si la fonction sera appelée si nous cliquons sur le bouton `Afficher`.

Si après avoir enregistré et cliqué sur le bouton rien ne s'affiche ou que vous ne voyez pas la fenêtre modale dans le navigateur, alors il y a une erreur.

Si vous pensez que cela ne marche pas, alors allez dans la console de votre navigateur. Si l'erreur provient du code JavaScript, dans ce cas, vous aurez une notification ou un message dans la console pour vous donner une indication concernant l'erreur.

Si tout est OK chez vous, vous pouvez enlever l'alerte dans la fonction `afficher_resultat` :

```
const afficher_resultat=()=>{  
  
}
```

Passons maintenant aux choses sérieuses!

1. Nous allons dans un premier temps récupérer ce que l'utilisateur a saisi et le sauvegarder pour une utilisation ultérieure:

```
const n=document.querySelector("#nombre").value;
```

Nous avons créé une variable `n` ( `const n` ) qui va nous permettre de stocker ce que l'utilisateur a saisi ( `document.querySelector("#nombre").value` ).

## Mais d'où vient le `#nombre` ?

Eh bien, si vous remarquez dans le code HTML, la zone de saisie a pour identifiant ( `id` ) `nombre`.

```
<input type="text" id="nombre" />
```

Donc, nous avons accédé à cette zone de saisie avec du JavaScript :

```
document.querySelector("#nombre");
```

En suite, nous avons récupéré ce qui est saisi avec l'attribut `.value` que nous voyons à la fin. Ce qui nous donne le code final :

```
const n=document.querySelector("#nombre").value;
```

2. Ensuite, nous allons vérifier si ce que l'utilisateur a saisi est bel et bien un nombre. Pour se faire, nous allons convertir ce que l'utilisateur a saisi en nombre entier.

La valeur contenue dans la variable `n` est pour le moment une chaîne de caractères. Voici comment nous pouvons vraiment convertir cette chaîne :

```
const n_entier=parseInt(n);
```

`parseInt( )` est une fonction native de JavaScript qui prend une chaîne de caractères en paramètre et renvoie sa forme convertie en entier.

S'il arrivait que l'utilisateur saisi autre chose qu'un chiffre, alors `parseInt( )` va renvoyer une valeur spéciale appelée `NaN` (Not a Number).

Pour vraiment savoir si l'utilisateur a bien saisi un chiffre, nous devons regarder si la valeur renvoyée par `parseInt()` n'est pas `NaN`. Voici comment nous pouvons faire cela:

```
if(isNaN(n_entier)){  
    alert("Vous devez saisir un nombre entier");  
    return;  
}
```

Décortiquons le code.

Déjà, nous avons l'utilisation de la condition `if` :

```
if(condition){  
    //code  
}
```

Le `if` nous permet d'exécuter le `code` lorsque la `condition` est vraie (`true`).

Donc la condition ici est `isNaN(n_entier)` qui vérifie si le nombre équivaut à `NaN` ou pas.

`isNaN` est aussi une fonction native de JavaScript.

Si `isNaN(n_entier)` est vraie (`true`) alors ce qui veut dire que l'utilisateur n'a pas saisi un chiffre.

Nous verrons ainsi l'alerte dans le navigateur.

Le `return` que nous voyons dans la condition `if` signifie tout simplement que nous allons sortir de la fonction et ne plus continuer l'exécution du code qui va suivre.

3. Nous allons créer maintenant le contenu que nous allons afficher dans le `div` qui a pour identifiant `resultat`.

```
<div id="resultat"></div>
```

Voici le code de la création du contenu :

```
let contenu="";
for(let i=0; i<=11; i++){
  contenu+=`${n}x${i} = ${n*i}<br />`;
}
```

Nous venons d'introduire ici une notion très capitale en JavaScript. Il s'agit bien évidemment de la boucle `for`.

Dans notre cas, nous partons d'une valeur initiale `0` avec `let i=0` en augmentant de `1` la valeur de `i` à chaque itération avec `i++` tout en vérifiant que nous ne dépassons pas `11` avec l'expression `i <=11`. D'où la notation :

```
for(let i=0; i<=11; i++){

}
```

À l'intérieur de cette boucle `for`, nous avons le code suivant :

```
contenu+=`${n}x${i} = ${n*i}<br />`;
```

Cette petite ligne de code renferme énormément de concept qui vous suivra partout dans vos codes JavaScript.

Déjà le `+` que nous voyons après `contenu` dans l'expression:

```
contenu+
```

Veut dire que nous sommes en train d'ajouter une information ou une valeur à la variable `contenu`. En terme technique, nous disons que nous faisons la `concaténation`.

En faisant la concaténation, nous gardons l'ancienne valeur qui est enregistrée. Nous devons le faire ainsi puisque nous sommes dans une boucle. Sinon, l'ancienne valeur sera écrasée à chaque nouvelle itération.



Ensuite, nous avons l'utilisation `backticks` qui est différent des simples côtes `' '` ou des griffes `" "`.

Sur un clavier de type `AZERTY`, vous pouvez obtenir le `backticks` avec la combinaison `alt gr + ``.

Les `backticks` sont utilisés à la place des simples côtes ou griffes si nous voulons afficher les variables JavaScript dans la chaîne ou même faire des calculs.

D'où l'écriture que nous voyons :

```
contenu+=`${n}x${i} = ${n*i}<br />`;
```

4. Il est temps maintenant d'afficher le résultat dans la page web au niveau du `div` qui a pour `id resultat`.

Voici le code qui va nous permet de faire cela :

```
document.querySelector("#resultat").innerHTML=contenu;
```

Une partie de ce code vous est peut-être familière. Nous voulons parler de cette partie :

```
document.querySelector("#resultat")
```

Cette partie du code est une référence du `div` qui a pour `id resultat` :

```
<div id="resultat"></div>
```

L'attribut `.innerHTML` est utilisé pour ajouter un contenu HTML à ce `div`, d'où le code final que vous voyez :

```
document.querySelector("#resultat").innerHTML=contenu;
```

Pour finir ce projet, nous allons implémenter l'évènement de `click` sur le bouton `Annuler` aussi comme nous l'avons fait pour le bouton `Afficher`.

```
<button onclick="annuler()">Annuler</button>
```

Créons ensuite la fonction `annuler()` dans notre fichier `script.js`

```
const annuler={()=>{
  document.querySelector("#resultat").innerHTML="";
  document.querySelector("#nombre").value="";
}}
```

Le rôle de cette fonction est simple :

- Vider la zone d'affichage du résultat
- Vider la zone de saisie

Voici le code entier du fichier script.js :

```
const afficher_resultat={()=>{
  let resultat=document.querySelector("#resultat");

  const n=document.querySelector("#nombre").value;
  const n_entier=parseInt(n);

  if(isNaN(n)){
    resultat.innerHTML="Vous devez saisir un nombre entier";
    return;
  }

  let contenu="";
  for(let i=0; i<=11; i++){
    contenu+=` ${n}x${i} = ${n*i}<br />`;
  }
  resultat.innerHTML=contenu;
}

const annuler={()=>{
  document.querySelector("#resultat").innerHTML="";
  document.querySelector("#nombre").value="";
}}
```

Nous venons de finir ce petit projet qui nous a permis de comprendre l'usage de JavaScript dans une page web.

## L'usage de la fonction native fetch

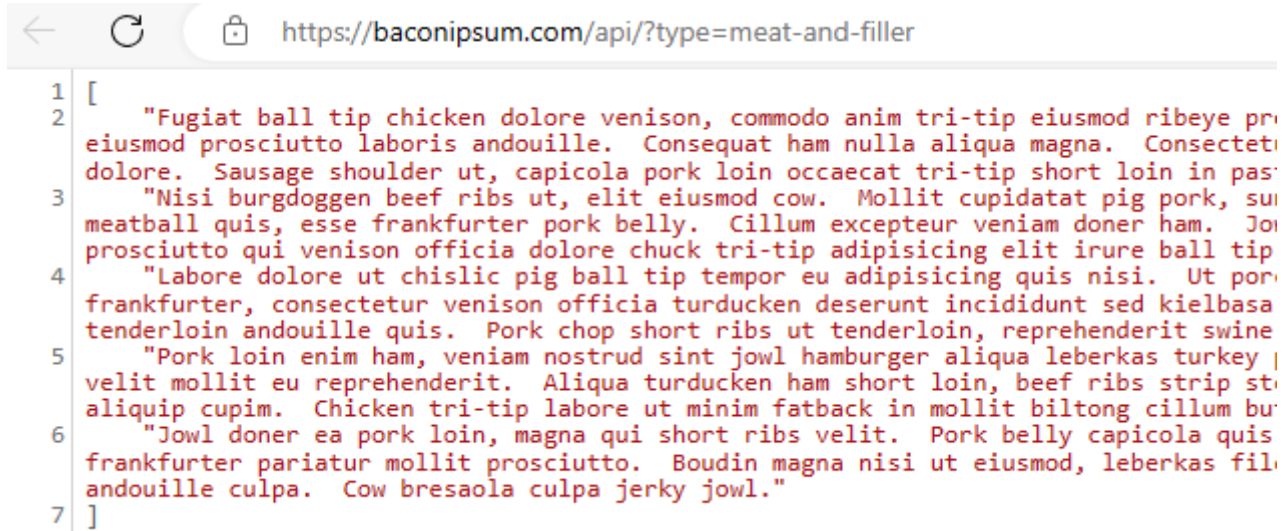
Il est primordiale de noter que nous avons la possibilité de faire appel aux services externes ou internes grâce à la fonction native `fetch()` de JavaScript. 😊

## Scénario

Nous voulons utiliser le service de texte aléatoire qui se trouve sur le lien suivant :

<https://baconipsum.com/api/?type=meat-and-filler>

Ouvrez ce lien dans un navigateur pour voir ce que renvoi ce service:



On peut voir que ce lien nous renvoi un contenu spécial sous un format `JSON`.

## Comment pouvons-nous récupérer ce contenu et l'utiliser dans notre application?

Voici comment nous pouvons le faire grâce à la fonction native `fetch()`. Nous allons créer une fonction JavaScript qui va s'appeler `recuperer_texte()` :

```
const recuperer_texte={()=>{
  }
}
```

Ceci est la façon traditionnelle de créer une fonction en JavaScript, mais nous allons apporter une petite modification à cette création de fonction :

```
const recuperer_texte=async ()=>{
  }
}
```

Nous avons ajouté le mot clé `async` dans la syntaxe de la création de la fonction, nous allons voir dans peu pourquoi nous avons fait cela.

Lançons maintenant une requête vers le service <https://baconipsum.com/api/?type=meat-and-filler> pour récupérer son contenu avec `fetch()` :

```
const recuperer_text=async ()=>{
  const reponse=fetch("https://baconipsum.com/api/?type=meat-and-filler");
}
```

Le rôle de `fetch` dans ce cas est de faire appel à l'URL qui lui est passé en paramètre notamment (<https://baconipsum.com/api/?type=meat-and-filler>) pour récupérer son contenu. Mais, ici cette requête peut prendre du temps (quelques secondes ou minutes) avant de nous renvoyer une réponse.

Comme nous avons besoin de la réponse de la requête pour continuer, nous devons attendre à ce que la requête nous renvoie une réponse.

Il faudra donc spécifier de façon explicite dans notre code que nous voulons attendre jusqu'à ce que la réponse soit disponible, nous allons légèrement modifier notre code en faisant ceci :

```
const recuperer_text=async ()=>{
  const reponse=await fetch("https://baconipsum.com/api/?type=meat-and-filler");
}
```

L'usage de `await` nous permet d'attendre la finition de la requête lancée par `fetch()`.

Une chose à retenir sur `await` est que son utilisation sera impossible si la fonction n'est pas créée avec le mot clé `async`. Voilà le pourquoi nous avons modifié légèrement la création de la fonction en utilisant le mot clé `async`.

Comme vous le savez déjà, <https://baconipsum.com/api/?type=meat-and-filler> nous renvoi un contenu de type `JSON`. La réponse obtenue doit être convertie sous un format JSON pour qu'on puisse l'utiliser dans notre application, voici comment nous pouvons effectuer cette conversion:

```
const contenu=await reponse.json();
```

La conversion peu parfois prendre du temps, raison pour laquelle nous devons utiliser `await`. La vraie conversion s'est faite en appliquant la fonction `json()` sur la réponse obtenue de `fetch`.

Nous pouvons maintenant utiliser le `contenu` qui est la plupart du temps un `tableau d'objet JSON`. Nous allons juste l'afficher dans la console avec le code suivant :

```
console.log(contenu);
```

Voici le code complet de la fonction `recuperer_text()` :

```
const recuperer_text=async ()=>{
  const reponse=await fetch("https://baconipsum.com/api/?type=meat-and-filler");
  const contenu=await reponse.json();
  console.log(contenu)
}
```

Voici un aperçu du résultat dans la console :

```
0: "Ut eiusmod cow proident, pig pork occaecat alcatra et ut commodo ham hock. Venison id incididunt
1: "Eu kevin tenderloin biltong turducken chuck. Dolore ut elit bresaola fugiat ground round. Flank
2: "Occaecat filet mignon sirloin excepteur lorem, consequat beef reprehenderit ut ea. Nostrud sint
3: "Corned beef chislic labore magna, nostrud shoulder sausage. T-bone ground round ipsum, sed spare
4: "Burgdoggen culpa nostrud fugiat ham hock picanha, short loin dolore eiusmod flank ad. Sunt jowl
length: 5
```

Comme vous pouvez le voir, le `contenu` est un tableau contenant 5 lignes. Nous allons essayer de parcourir ce tableau avec la fonction native `map()` de JavaScript pour afficher chaque ligne :

```
contenu.map((ligne)=>{
  console.log(ligne);
})
```

La fonction `map()` est très importante dans JavaScript et elle est très utilisée dans la manipulation des tableaux en JavaScript.

Il faut faire une recherche sur cette fonction et aussi son demi-frère `filter()` qui est, lui aussi, une fonction native de JavaScript utilisée dans la manipulation des tableaux.

## Plus loin avec JavaScript

Si votre objectif est de devenir développeur web, vous devez faire un pas en avant en adoptant un Framework front-end JavaScript pour améliorer votre productivité et être compétitif dans le monde professionnel.

Parmi ces Framework, les plus populaires de nos jours sont :

- ReactJS
- VueJS
- AngularJS

Nous vous conseillons de commencer avec `ReactJS` à cause de sa popularité et son attachement très fort avec le JavaScript.

Un chapitre est consacré à une introduction du Framework `ReactJS` pour vous aider à améliorer votre performance en tant que développeur web.

# Back-End

Le back-end, contrairement aux front-end, est tout ce que nous ne voyons pas dans le navigateur, mais nécessaire pour la dynamisation de notre application web.

De nos jours, nous avons deux types de back-end :

- Server-based back-end
- Serverless back-end

Dans le `server-based` backend, nous pouvons utiliser les langages qui ne s'exécutent que sur un serveur comme PHP, nodeJS, JAVA, Python, etc.

Nous allons utiliser dans ce livre `nodeJS` puisqu'il est basé sur JavaScript dont vous avez déjà une petite notion. Vous pouvez apprendre par après le `PHP` qui est aussi très utilisé à cause de sa simplicité pour la mise en production.

Au niveau du `Serverless`, nous allons utiliser des services du cloud pour héberger ou mettre à disposition du grand public notre application au lieu de les gérer par notre propre serveur. Nous pouvons dans ce cas utiliser les services d'Amazone(AWS) ou de Google (Firebase) pour programmer notre backend de type `Serverless`. La bonne nouvelle pour vous au niveau du `Serverless` est que, nous pouvons faire tout ceci à partir du simple JavaScript.

Vous pouvez remarquer que maîtriser JavaScript vous sera grandement utile dans votre parcours dans le développement web.

## NodeJS



Avec `nodeJS` qui est un langage côté serveur, nous pouvons :

- Sauvegarder les données utilisateur dans une base de données,
- Envoyer des fichiers sur le serveur,
- Créer des sessions utilisateur,
- Envoyer des mails ,
- Etc.

Il est très rare de créer une application web qui ne nécessite pas de backend. Il est primordial pour vous d'acquérir des connaissances dans le backend.

## Installation de nodeJS

Ouvrez votre invite de commande et tapez la commande suivante :

```
node --version
```

Si vous avez un retour pareil à la figure ci-dessous, alors nodeJS est déjà installé sur votre machine

```
v16.13.1
```

Si par contre, vous avez un message d'erreur dans le terminal disant que la commande est introuvable, donc vous n'avez pas encore nodeJS sur votre ordinateur.

Allez dans les [annexes](#) pour suivre l'[installation de nodeJS](#) avant de continuer.

## Scenario

Vous avez une application web qui permet aux utilisateurs de renseigner leur nom, email et téléphone. Après validation, l'interface client (front-end) envoie ces informations au serveur (back-end) pour les sauvegarder dans un fichier nommé `users.txt` qui est sur le serveur.

Voici un aperçu de ce que nous allons construire comme application

**Inscrivez-vous maintenant**

Votre nom

Votre mail

Votre téléphone

**Valider**

Nous allons apprendre nodeJS à travers ce scénario que nous venons de décrire.

## Construction du front-end

Nous allons essayer de mettre en place le front-end de cette application et ensuite le connecter au back-end que nous allons voir dans peu 😊

Créons un dossier pour ce projet, nous allons le nommer `inscription`. Ensuite, nous allons créer les 3 fichiers standard de notre application dans le dossier `inscription` :

- `index.html`
- `style.css`
- `script.js`

Voici le contenu du fichier `index.html`

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inscrivez-vous</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>
  <body>
    <div id="form">
      <h1>Inscrivez-vous maintenant</h1>
      <div class="info">
```

```

    <label>Votre nom</label>
    <input type="text" id="nom" />
</div>

<div class="info">
    <label>Votre mail</label>
    <input type="email" id="email" />
</div>

<div class="info">
    <label>Votre téléphone</label>
    <input type="tel" id="telephone" />
</div>

<div class="info">
    <button>Valider</button>
</div>

</div>
</body>
</html>

```

Nous avons été très minimaliste dans ce fichier pour ne pas trop compliquer l'interface client. Voici maintenant le contenu du fichier css :

```

body{
    display:flex;
    justify-content:center;
}
#form{
    border:1px solid silver;
    width:300px;
    margin-top:2rem;
    padding:1rem;
    border-radius:10px;
    background-color:#efefef;
}

#form h1{
    text-align:center;
    text-decoration:underline;
    font-size:20px;
}

#form .info{

```

```

display:flex;
flex-direction:column;
margin-bottom:1rem;
}

#form .info label{
  font-size:14px;
}

#form .info input{
  width:250px;
  padding:0.5rem;
  outline:none;
  border:1px solid silver;
  border-radius:5px;
  color:gray;
}

#form .info button{
  width:125px;
  padding:0.5rem;
  background-color:green;
  border:none;
  border-radius:5px;
  color:white;
  cursor:pointer;
  font-weight:bold;
}

```

Actuellement, le contenu du fichier script.js est vides.

Maintenant que l'interface utilisateur (front-end) est en place, nous allons regarder l'action de validation qui fera appel au serveur.

Selon notre scénario, lorsque l'utilisateur clique sur le bouton `valider`, nous devons récupérer les informations saisies par l'utilisateur et les envoyer au back-end que nous allons mettre en place dans peu.

Essayons quand même de préparer le front-end pour cette action d'envoi d'informations vers le back-end.

Nous devons pour ce faire, ajouter l'attribut `onclick` au bouton `valider` et lui donner comme valeur une fonction que nous allons nommer `enregistrer()` qui doit être créée dans le fichier `script.js`.

```
<button onclick="enregistrer()">Valider</button>
```

Allons maintenant créer la fonction `enregistrer()` dans le fichier `script.js`

```
const enregistrer=async ()=>{  
  const nom=document.querySelector("#nom").value;  
  const email=document.querySelector("#email").value;  
  const telephone=document.querySelector("#telephone").value;  
}
```

Comme vous pouvez le voir, la fonction `enregistrer` a pu récupérer les données et est désormais prête à effectuer l'envoi de ces données aux serveurs. 😎

Mais comment pouvons-nous envoyer ces données au serveur ? 😞

Eh bien, nous allons nous servir de la fonction native `fetch()` de JavaScript pour le faire.

Mais bien avant, il faut qu'on sache sous quel format on va envoyer les données au serveur. 😬  
C'est très indispensable de savoir le format des données d'avance parce que le serveur doit s'avoir quel type ou format de données il va recevoir pour mettre en place les bonnes techniques de réception de ces données.

Nous avons parlé du format `JSON` entre temps dans ce livre et nous espérons que vous avez fait des recherches pour en savoir plus. Nous allons utiliser ce format pour envoyer nos données vers le serveur.

Ce travail va se faire en deux temps:

- Former un objet javascript

```
const obj={nom,email,telephone}
```

- Convertir l'objet vers le format JSON

```
const obj_json=JSON.stringify(obj);
```

Voici le code actuel de la fonction `enregistrer()` :

```
const enregistrer=async ()=>{  
  const nom=document.querySelector("#nom").value;  
  const email=document.querySelector("#email").value;  
  const telephone=document.querySelector("#telephone").value;
```

```
const obj={nom,email,telephone};
const obj_json=JSON.stringify(obj);
}
```

Maintenant que nous avons notre donnée formatée sous le format `JSON`, nous pouvons maintenant l'envoyer au serveur avec la fonction native `fetch()`.

```
const requete=await fetch("http://localhost:5000/enregistrement",{
method:"POST",
body:obj_json,
headers:{
  "Content-Type":"application/json"
}
})
let reponse=await res.text();
console.log(reponse)
```

Mais c'est quoi le lien `http://localhost:5000/enregistrement` que nous avons dans le `fetch()` ?



Retenez tout simplement ici que ceci représente l'adresse de notre serveur que nous allons bientôt construire. Vous allez mieux comprendre dans la suite pourquoi nous avons mis cela.



Pour tester l'application, il faut la lancer avec `live-server` ou tout autre logiciel serveur sinon le `fetch` ne marchera pas. Allez dans les annexes pour suivre l'installation de `live-server` et comment vous pouvez lancer l'application web avec.

Maintenant essayons de tester l'application dans le navigateur. Une fois l'application lancée avec `live-server`, vous devez voir l'adresse `127.0.0.1:8080` ou `localhost:8080`. Ceci représente l'URL ou l'adresse de l'application front-end:

**Inscrivez-vous maintenant**

Votre nom

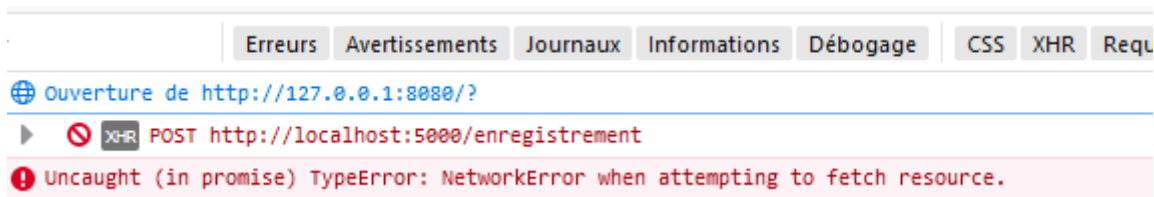
Votre mail

Votre téléphone

**Valider**

Essayons de renseigner les champs et cliquons sur le bouton `valider`.

Une fois valider, nous aurons une erreur dans la console du navigateur 😬 :



Cette erreur signale que notre application n'arrive pas à avoir accès à l'URL `http://localhost:5000/enregistrement`. Ce qui est un peu normal, puisque, nous avons dit, ce lien représente l'URL du serveur, et nous n'avons pas encore construit le serveur.

Essayons de régler ce problème en construisant maintenant le serveur. 😊

## Initialisation du serveur

La première chose à faire est d'ouvrir l'invite de commande dans le dossier de notre projet et saisir la commande suivante :

```
npm init -y
```

Si vous n'avez pas installé `nodeJs` sur votre ordinateur, la commande `npm` ne marchera pas. Allez dans les annexes pour suivre l'installation de `nodeJs` si ce n'est pas encore fait.

Cette commande va initialiser le fichier de configuration `package.json` à l'intérieur de notre dossier `inscription`

Voici à quoi peu ressemble le contenu du fichier `package.json` , cela peut être légèrement différent sur votre ordinateur :

```
{
  "name": "inscription",
  "version": "1.0.0",
  "description": "",
  "main": "script.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Ce contenu nous donne beaucoup d'informations sur notre serveur, mais nous allons intéresser spécialement l'attribut `main` et `script`.

## L'attribut main

Cet attribut a pour valeur le nom du fichier de notre serveur. Actuellement, on voit que cette valeur est `script.js`.

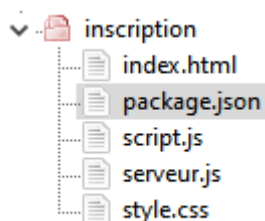
```
"main": "script.js",
```

Mais, nous avons déjà un fichier `script.js` qui est lié à notre front-end. C'est pour cela que nous allons modifier la valeur de l'attribut `name` et mettre `serveur.js` à la place.👉

```
"main": "serveur.js",
```

Il nous faut donc créer ce fichier à l'intérieur du dossier `inscription`

Voici ci-dessous, le contenu actuel de notre dossier `inscription`





## L'attribut scripts

Cet attribut va nous permettre de mettre en place la commande de lancement de notre serveur.



En effet, il faut spécifier dans le fichier de configuration `package.json` dans l'attribut `scripts` comment est-ce que le serveur doit être lancé.

Voici ce que nous devons mettre à l'intérieur de cet attribut :

```
"start": "node serveur.js"
```



Ce code indique tout simplement que notre fichier `serveur.js` sera exécuté par le logiciel `nodejs` quand on saisira la commande `npm start`.

Voici en gros, le contenu actuel de notre fichier `package.json` après toutes ces modifications :

```
{
  "name": "inscription",
  "version": "1.0.0",
  "description": "",
  "main": "serveur.js",
  "scripts": {
    "start": "node serveur.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

## Création du serveur avec express

Dans cette section, nous allons vraiment créer le serveur. Nous allons nous faire aider par un module appelé `express`.

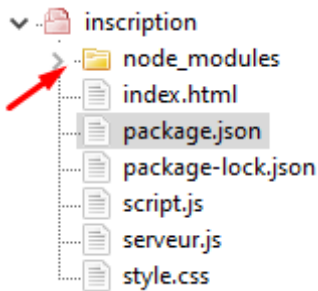
Il faut déjà commencer à installer ce module dans notre projet `inscription`. Nous allons pour ce faire, lancer la commande suivante dans notre dossier `inscription`:

```
npm install express
```

Après quelques secondes, `express` sera installé dans notre projet et prêt à être utilisé. Mais avant de commencer la création du serveur, nous devons expliquer quelques changements qui

se sont opérés après l'installation de `express` dans notre projet.

Déjà, nous avons l'apparition d'un dossier qui se nomme `node_modules` qui va renfermer le code source de tous les outils qu'on aura à installer et d'autres outils qui sont nécessaires pour le bon fonctionnement du serveur.



Nous remarquons aussi que le fichier `package.json` a subi une modification avec l'apparition d'un autre attribut du nom de `dependencies`, voici son contenu actuel :

```
{
  "name": "inscription",
  "version": "1.0.0",
  "description": "",
  "main": "serveur.js",
  "scripts": {
    "start": "node serveur.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

L'attribut `dependencies` va regrouper tous les modules qu'on aura à installer, actuellement, nous avons seulement installé `express` et c'est ce que nous voyons dans cette partie.

La valeur `4.18.1` que nous voyons représente la version du module `express` que nous avons installé.

Ouvrons maintenant le fichier `serveur.js` et mettons-y le code de création du serveur :

```
const express=require("express");
const app=express();
app.listen(5000);
```

Trois lignes de code, c'est tout ce qu'il faut pour créer notre serveur. Mais une explication s'impose.

```
const express=require("express");
```

Ce code fait appel à notre module `express` que nous avons installé pour le rendre disponible dans le fichier `serveur.js`.

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

Ce code ci-dessus crée le serveur en question avec le module `express`.

Une fois que le serveur est créé, nous devons le mettre en écoute sur un `port` donné pour que le client puisse l'accéder. C'est ce que fait le code suivant :

```
app.listen(5000)
```

Le serveur écoute sur le port `5000` comme vous pouvez le voir. Vous pouvez changer ce `port` si vous voulez.

## Lancer le serveur

Une fois que le serveur est bien créé, nous devons le lancer. Pour cela, nous allons lancer la commande suivante :

```
npm start
```

Mais d'où vient cette commande ?

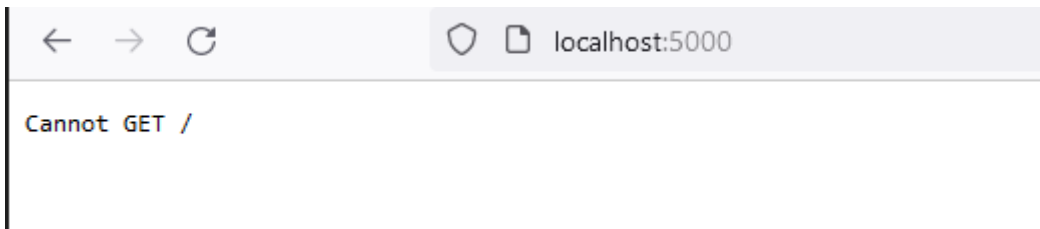
Cette commande provient de notre fichier `package.json` dans la section `scripts`.

```
"scripts": {  
  "start": "node serveur.js"  
}
```

En fait, cette commande va faire appel à `nodeJS` qui est installé sur votre ordinateur et exécuter le contenu du fichier `serveur.js`.

Une fois que la commande est bien lancée, nous pouvons aller dans notre navigateur et accéder au serveur avec le lien suivant :

```
127.0.0.1:5000 == ou http://localhost:5000
```



## Récapitulatif

Donc en résumé, notre front est lancé sur l'adresse `http://localhost:8080` et notre serveur est lancé sur le l'adresse `http://localhost:5000`.

Assurez-vous que les deux parties (front-end et back-end) sont bien accessibles sur leur URL respectives.

## Cannot GET /

Si nous essayons d'accéder au serveur sur son URL `http://localhost:5000`, nous avons ce message qui est affiché sur la page web.

Ce message veut tout simplement dire que le chemin `/` n'existe pas sur le serveur avec la méthode `GET`.

Essayons de créer ce chemin dans notre fichier `serveur.js`

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

app.get("/",(requete,reponse)=>{
  reponse.send("Bonjour tout le monde")
})

app.listen(5000);
```

Nous venons de modifier le contenu du fichier `serveur.js` avec la création du chemin `/` utilisant la méthode `GET`.

Le paramètre `requete` représente la demande de la partie client (front-end) et le paramètre `reponse` est le contenu de la réponse du serveur.

Maintenant, nous allons arrêter le serveur ( `CRTL + c` ) et relancer de nouveau avec la commande `npm start`.

Si nous allons encore dans le navigateur après cette modification, on peut voir désormais le contenu du serveur qui doit nous afficher `Bonjour tout le monde`.

Mais selon notre application front-end, on a besoin du chemin `/enregistrement` avec la méthode `POST` pour l'envoi des données vers le serveur:

```
await fetch("http://localhost:5000/enregistrement",{
  method:"POST",
  body:obj_json,
  headers:{
    "Content-Type":"application/json"
  }
})
```

Vous vous rappelez de ce code qui se trouve dans votre fichier `script.js` n'est-ce pas ? 👍

Et ce code doit avoir un peu de sens maintenant si vous avez bien suivi jusqu'ici.  
En effet, on retrouve dans `http://localhost:5000/enregistrement` l'adresse du serveur (`http://localhost:5000`) et le chemin (`/enregistrement`) qui doit être créé dans notre fichier `serveur.js` avec la méthode `POST` mentionné dans la requête `fetch`.

Voici comment nous pouvons créer ce chemin:

```
app.post("/enregistrement",(requete,reponse)=>{
  reponse.send("Reponse du serveur")
})
```

Vous remarquez que c'est pratiquement la même chose que ce que nous avons vu au niveau de la création du chemin `/`, sauf que ici, nous avons utilisé `post` au lieu de `get` puisque `POST` est utilisé dans l'attribut `method` de notre fonction `fetch()`.

## CORS Missing Allow Origin

Si nous essayons à nouveau de cliquer sur le bouton `valider` dans le front-end, nous avons une erreur de ce genre :

```
CORS Missing Allow Origin
Blocage d'une requête multiorigines (Cross-Origin Request) : la politique « Same Origin » ne permet pas de consulter la ressource distante située sur
http://localhost:5000/enregistrement. Raison : l'en-tête CORS « Access-Control-Allow-Origin » est manquant. Code d'état : 404.
```

Ce message veut tout simplement dire que le client est lancé sur une adresse différente de celle du serveur. En fait notre navigateur essaie de nous protéger en quelque sorte. 😬

Pour régler ce problème, nous allons dire au serveur d'accepter toute requête venant de n'importe quelle adresse. 😞

Nous allons le faire grâce un module qui s'appelle `cors`. Installons ce module avec la commande suivante :

```
npm install cors
```

Ensuite nous pouvons faire appel à ce module avec le code suivant :

```
const cors=require("cors");
```

Enfin nous pouvons l'utiliser de la manière suivante :

```
app.use(cors());
```

Voici le code complet du fichier serveur.js

```
const express=require("express");
const app=express();
const cors=require("cors")

app.use(cors());

app.post("/enregistrement",(requete,reponse)=>{
  reponse.send("Reponse du serveur")
})

app.listen(5000);
```

Maintenant si nous relançons de nouveau l'application, nous devons avoir le message `Réponse du serveur` dans la console du serveur.

## Récupération des données envoyées par le front-end

Le serveur est presque en mesure de récupérer les données envoyées par le client (front-end). Nous avons spécifié dans la requête `fetch` que nous sommes en train d'envoyer des informations de type `JSON` vers le serveur.

C'est le pourquoi nous devons ajouter la ligne suivante à notre serveur pour qu'il soit capable de travailler avec les données sous format `JSON` :

```
app.use(express.json());
```

Ce code doit être ajouté au serveur avant la code de création du chemin `/enregistrement`.

Maintenant nous pouvons récupérer les données envoyées par le client de la façon suivante :

```
app.post("/enregistrement", (requete, reponse) => {  
  const data = requete.body;  
  reponse.send("Reponse du serveur")  
})
```

Vous vous rappelez de `body` dans notre requête `fetch` ? 😊

Eh bien, c'est exactement ce que nous venons de récupérer ici avec le code suivant :

```
const data = requete.body; 🧐
```

La variable `data` est donc un objet JavaScript composé des attributs `nom`, `email`, `telephone`. Donc, nous pouvons `déstructurer` la variable `data` avec le code suivant:

```
const {nom, email, telephone} = data;
```

Le code actuel du chemin `/enregistrement` est :

```
app.post("/enregistrement", (requete, reponse) => {  
  const data = requete.body;  
  const {nom, email, telephone} = data;  
  reponse.send("Reponse du serveur")  
})
```

## Enregistrement des données dans un fichier

Il est maintenant temps qu'on enregistre ces données (`nom`, `email`, `telephone`) dans un fichier qui va s'appeler `users.txt` sur le serveur.

Nous avons besoin du module de gestion de fichier `fs`. Ce module est natif à `nodeJS` et nous n'avons pas besoin de l'installer avec `npm install fs`. Il nous faut juste l'importer dans le fichier du serveur comme suit :

```
const fs=require("fs");
```

Nous pouvons maintenant ajouter les données au fichiers `users.txt` avec le code suivant :

```
fs.appendFile("users.txt",`${nom} ${email} ${telephone} \n`,()=>{});
```

Voici le code complet du fichier `serveur.js` :

```
const express=require("express");
const app=express();
const cors=require("cors");
const fs=require("fs");

app.use(cors());
app.use(express.json());

app.post("/enregistrement",(requete,reponse)=>{
  const data=requete.body;
  const {nom,email,telephone}=data;
  fs.appendFile("users.txt",`${nom} ${email} ${telephone} \n`,()=>{});
  reponse.send("Reponse du serveur")
})

app.listen(5000);
```

Essayons de rentrer les données comme sur la figure ci-dessous dans le front-end :

**Inscrivez-vous maintenant**

Votre nom










Votre mail

Votre téléphone

Valider



Après validation, on peut voir dans le dossier `inscription` un nouveau fichier `users.txt` qui est créé.

 <code>node_modules</code>	25/11/2022 14:24	Dossier de fichiers	
 <code>index</code>	25/11/2022 08:50	Firefox HTML Doc...	1 Ko
 <code>package</code>	25/11/2022 14:24	Fichier source JSON	1 Ko
 <code>package-lock</code>	25/11/2022 14:24	Fichier source JSON	41 Ko
 <code>script</code>	25/11/2022 14:27	Fichier de JavaScript	1 Ko
 <code>serveur</code>	25/11/2022 15:38	Fichier de JavaScript	1 Ko
 <code>style</code>	25/11/2022 08:50	Document de feui...	1 Ko
 <code>users</code> 	25/11/2022 15:39	Document texte	1 Ko

Si vous ouvrez ce fichier, vous verrez le contenu suivant:

```
BOB bob@gmail.com 92929562
```

Vous pouvez continuer et ajouter d'autres données en remplissant le formulaire de nouveau.

# ReactJS

Comme vous le s'avez déjà, ce chapitre va couvrir `ReactJS` pour le développement des applications web.

## Mais c'est quoi ReactJS ? 🤔

Pour ne pas vous surcharger avec trop d'histoire sur `ReactJS`, notez que `ReactJS` est une `librairie` créée par Facebook pour faciliter le développement de la partie front-end des applications web avec la technique de création des composants réutilisable. 😊

Si le mot `librairie` en informatique vous est nouveau, voyez-la tout simplement comme une boîte dans laquelle se trouve des outils que vous pouvez utiliser pour faire votre travail. 😊

Tout ceci deviendra de plus en plus clair dans la suite de ce livre. Ne paniquez pas, surtout si vous ne comprenez pas tout au début. Tout sera clair bientôt pour vous. Un peu de patience. 😊

## Pourquoi ReactJS ?

`ReactJS` nous offre d'énormes avantages dans le développement web. Voici quelques avantages qu'offre `ReactJS` :

- Il vous permet de créer des composants réutilisables et qui maintiennent leur propre état.
- Il vous permet de faciliter la création d'interface utilisateur interactive et rend la réactualisation de la page plus rapide.
- Il est probable que vous vous intéressiez par la suite au développement d'applications mobiles, une maîtrise de `ReactJS` vous permet de faire une transition en douce vers la librairie `React-Native` qui est utilisée pour le développement des applications mobiles. 🙌

Avant de commencer à créer nos premiers projets avec `ReactJS`, nous devons installer NodeJS sur notre ordinateur.

## Installation de NodeJS

Avant toute chose, nous devons avoir `NodeJS` sur notre ordinateur. Pour vérifier si `NodeJS` est présent sur votre ordinateur, vous devez lancer l'invite de commande et saisir le code suivant :

```
node --version
```

Si `NodeJS` est présent sur votre ordinateur, vous devez avoir des informations dans la console, un

peu comme dans la capture suivante :

```
C:\Users\FOGAN>node --version  
v16.13.1
```

Si vous avez une erreur, alors `nodeJS` n'est pas installé, allez donc dans les annexes et voir le chapitre consacré à l'installation de `NodeJS` avant de continuer.

## Création d'un projet ReactJS

Nous allons commencer à rentrer dans le cœur de ReactJS. Ce chapitre va vous montrer comment

créer un nouveau projet avec `ReactJS`.

Après la création de ce premier projet, nous allons profiter de l'occasion pour vous expliquer le rôle que joue chaque fichier et dossier générés dans le processus de création d'un nouveau projet ReactJS.

## L'outil create-react-app

`Create-react-app` est un outil développé par la communauté `ReactJS` pour faciliter la mise en place d'un nouveau projet `ReactJS`.

Installons donc `create-react-app` avec la ligne de commande suivante :

```
npm install -g create-react-app
```

Ouvrez votre invite de commande et placez-vous à l'emplacement dans lequel vous voulez créer

votre projet. Dans notre cas, nous avons créé un dossier nommé `projet React` sur le bureau de notre ordinateur. Et nous allons accéder à ce dossier dans l'invite de commande comme suit :

```
H:\OneDrive\Bureau\projet react>
```

Maintenant, tapons la commande suivante :

```
npx create-react-app projet1
```

Dans cette commande, `projet1` représente le nom de notre projet. Vous êtes libre de choisir le nom que vous voulez pour votre projet. Mais, pour des raisons de simplicité et pour bien suivre ce que nous faisons, nous vous conseillons de mettre le même nom `projet1`.

Valider ensuite avec la touche entrée pour commencer la création de votre nouveau projet.

Après quelques secondes, et si tout s'est bien passé, vous devez avoir un contenu dans l'invite de commande comme suit :

```
npm start
  Starts the development server.

npm run build
  Bundles the app into static files for production.

npm test
  Starts the test runner.

npm run eject
  Removes this tool and copies build dependencies, configuration files
  and scripts into the app directory. If you do this, you can't go
  back to using create-react-app.

We suggest that you begin by typing:

  cd projet1
  npm start

Happy hacking!
```

Si vous êtes arrivé à cette étape alors bravo!!! 😎

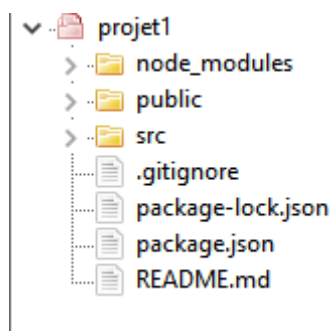
Vous venez juste de créer votre premier projet avec `ReactJS`.

NB: Il est aussi possible d'utiliser l'outil `Vite` au lieu de `create-react-app` pour générer nos projets `reactJs` :

```
npm create vite projet1
```

## Lancer le projet

Retournez sur le bureau de votre ordinateur dans le dossier `projet react`. Vous devez voir un nouveau dossier nommé `projet1`. Ce dossier est créé par l'outil `create-react-app`.



Rappelez-vous que `projet1` est le nom que nous avons saisi lors de la création du projet.

Avant de parcourir le contenu du dossier de notre projet, nous allons essayer de lancer le projet nouvellement créé.

Pour se faire, accédons dans l'invite de commande notre dossier `projet1` en faisant :

```
cd projet1
```

Nous allons nous retrouver dans le dossier `projet1` et ensuite tapons la commande suivante pour lancer le projet :

```
npm start
```

Après validation de cette commande avec la touche entrée, nous allons voir dans la console un message nous signalant que notre projet est en cours démarrage.

Quelques seconds plus tard, le projet sera lancé dans un navigateur web sur notre ordinateur à l'adresse `http://localhost:3000`



Ceci est le contenu par défaut qui est généré par l'outil `create-react-app`. Nous allons très bientôt

voir comment nous pouvons changer le contenu de cette page.

NB: notre invite de commande a cette page ressemble à ceci :

```
Compiled successfully!

You can now view projet1 in the browser.

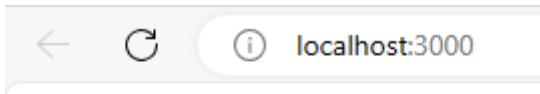
Local:      http://localhost:3000
```

Ce qui veut dire que notre page web située à l'adresse `http://localhost:3000` est directement liée à notre invite de commande qui lui donne l'autorisation de fonctionner sur le port `3000`. Si nous fermons l'invite de commande alors notre page web ne peut plus répondre.

Vous pouvez si besoin arrêter le serveur dans l'invite de commande en utilisant la combinaison de touche `Ctrl+c`.

## contenu de notre nouveau projet généré

Voici à quoi ressemble notre dossier `projet1` actuellement :



## Je suis une personne



Parcourons ces différents dossiers et fichiers pour comprendre un peu ce qui se passe ici.

Dans la racine du projet, nous avons quelques fichiers de configurations :

- Le fichier `package-lock.json` contient les informations sur la version des modules que nous utilisons. Nous n'allons pas toucher au contenu de ce fichier,
- Le fichier `package.json` regroupe les modules généraux que notre projet utilise. On peut localiser ces modules dans l'attribut `dependencies` de ce fichier.

```
"dependencies": {
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-scripts": "5.0.1",
  "web-vitals": "^2.1.4"
}
```

Nous pouvons voir dans ce même fichier, l'ensemble des commandes à exécuter dans l'attribut `scripts` :

```
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "test": "react-scripts test",  
  "eject": "react-scripts eject"  
}
```

Vous vous rappelez qu'on avait utilisé `npm start` pour lancer le projet ? Eh bien, c'est justement parce que l'attribut `scripts` de ce fichier contient un `start`.

Nous allons plus tard utiliser `npm run build` pour avoir la version de notre application web qui doit être hébergée.

- Le dossier `node_modules` renferme le code source de tous les modules de notre application. Nous n'allons jamais ajouter ou modifier manuellement le contenu de ce dossier,
- Le dossier `public` est davantage intéressant dans la mesure où nous y avons un fichier `index.html` de notre page qui renferme un `div` avec un attribut `id` `root`.

L'ensemble des contenus que nous allons créer plus tard sera automatiquement injecté dans ce `div`.

Nous n'allons pas modifier ce fichier. Il restera aussi intact tout au long de la réalisation de notre application.

Toutefois, nous pouvons ajouter d'autres librairies comme `bootstrap` ou `tailwindcss` à ce fichier si nous voulons bien dans la balise `head`.

- Le dossier qui nous intéresse vraiment est le dossier `src`. Le fichier principal de ce dossier est `index.js` ou `main.js` si vous aviez créé le projet avec `vite` au lieu de `create-react-app`.

Le code contenu dans ce fichier peut nous paraître pas clair pour le moment. Mais, si vous regardez bien sur la ligne suivante :

```
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>  
);
```



Ce code injecte le composant `App` dans le `div` qui a pour id `root` défini dans le fichier `index.html` du dossier `public`. 😞

Le composant `App` est englobé par le conteneur `React.StrictMode`, qui nous aide dans le développement de l'application en nous obligeant à être stricte dans le codage. Ce conteneur peut être enlevé.

Le composant `App` est importé avec la ligne suivante :

```
import App from './App';
```

Ce qui veut dire que le fichier `App.js` nous a créé un composant `App` que nous venons d'ajouter à notre `div` qui a pour id `root`.

Et si nous ouvrons le fichier `App.js`, on voit le contenu actuel qui est affiché sur notre projet web.

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a className="App-link" href="https://reactjs.org" target="_blank" rel="noopener
noreferrer">
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```



Nous allons revenir dans la section suivante dans ce fichier.

Mais pour le moment, essayons d'enlever l'ensemble du code contenu dans le `div` qui a pour `className` la valeur `App` et mettons le texte `Bonjour` dans la balise `h2` :

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <h2>Bonjour</h2>
    </div>
  );
}

export default App;
```



Enregistrez ce fichier avec la combinaison de touches `Ctrl+s`. Vous devez remarquer que notre page web est automatiquement actualisée.

Nous pouvons supprimer le fichier `logo.svg` puisque nous ne l'utilisons pas.

Voici ensuite le nouveau contenu du fichier `App.js` :

```
import './App.css';

function App() {
  return (
    <div className="App">
      <h2>Bonjour</h2>
    </div>
  );
}

export default App;
```



En react, chaque composant que nous créons peut avoir un fichier `css` pour ajouter du style à ce

composant. C'est la raison pour laquelle pour le composant `App.js`, nous avons un autre fichier du même nom `App.css` qui est d'ailleurs importé dans `App.js`.

Nous allons revenir sur le fichier `css` dans une section réservée à l'application des styles sur nos

composants vers la fin de ce chapitre.

Enlevons tout le code contenu dans le fichier `App.css` et gardons seulement le code suivant :

```
.App {  
  
}
```

Parce que dans notre application `App.js`, nous avons seulement une classe nommée `App`.

C'est toujours dans la même logique que nous avons le fichier `index.css` qui définit certains styles globaux pour notre fichier `index.js`.

Nous allons revenir sur le fichier `App.test.js` et `registerServiceWorker.js` plus tard dans ce livre. Mais notez que nous n'allons pas les toucher.

Nous avons commencé à dire que `App.js` est un composant. Nous allons modifier le contenu de notre application en modifiant le contenu de `App.js` ou en y ajoutant d'autres composants créés.

Nous allons pour se faire analyser le contenu de `App.js` dans la section suivante pour comprendre ce qui se passe et pourquoi nous l'appelons un composant. 😊

## Contenu du fichier App.js

Avant de continuer, nous allons faire un peu de ménage dans le fichier `App.js`

Voici le nouveau contenu du fichier `App.js`

```
import './App.css';  
  
function App() {  
  return (  
    <div className="App">  
      <h2>Bonjour</h2>  
    </div>  
  );  
}  
  
export default App;
```

Avec ce changement, vous verrez instantanément dans le navigateur web le texte `Bonjour`.

```
import './App.css';
```

Ce code fait appel au fichier `CSS` qui sera utilisé pour mettre en forme le composant qui est créé dans ce fichier.

Nous avons ensuite une fonction JavaScript qui prend le nom du fichier :

```
function App(){  
  
}
```

Nous pouvons utiliser la syntaxe moderne avec le code suivant :

```
const App={()=>{  
  
}}
```

Au sein de cette fonction, nous avons le mot clé `return` qui renvoie un contenu `HTML`. Ce contenu `HTML` représente le composant qui est créé.

```
return (  
  <div className="App">  
    <h2>Bonjour</h2>  
  </div>  
);
```

Le contenu renvoyé n'est rien d'autre que du `HTML`. Il faut quand même noter que nous ne pouvons que mettre un seul composant dans la parenthèse du `return`.

Actuellement, nous avons :

```
<div className="App">  
  <h2>Bonjour</h2>  
</div>
```

Ceci représente un seul composant `div` qui est renvoyé. Cependant, le seul composant peut contenir d'autres composants enfants.

Par contre, nous ne pouvons pas faire ceci dans la parenthèse du `return` :

```
return (  
  <div className="App">  
    <h2>Bonjour</h2>  
  </div>  
  <h3>Un autre contenu</h3>  
)  
);
```

Sur la dernière ligne, nous avons ici :

```
export default App;
```

C'est grâce à cette dernière ligne que nous pouvons utiliser le composant `App` dans un autre fichier.

Le fichier `App.js` est considéré comme le conteneur de notre application. C'est donc dans le composant `App` que nous allons héberger tous les autres composants de notre application.

## Création d'un composant

Comme nous l'avons dit, un composant n'est rien d'autre qu'une fonction JavaScript renvoyant un contenu HTML. 👍

Par convention, chaque composant doit avoir son propre fichier JavaScript dans lequel il est créé et exporté.

### Application :

Créons un composant nommé `Personne` qui affiche le texte "je suis une personne" dans le fichier `App.js`

### Solution :

Nous allons commencer à créer le fichier `Personne.js` dans le dossier `src` au même endroit que le fichier `App.js` et qui va contenir le code de création du composant `Personne` comme suit :

```
const Personne=()=>{  
  return(  
    <h2>Je suis une personne</h2>  
  )  
}  
export default Personne;
```

Nous venons de créer le composant `Personne`. Nous utilisons par convention une lettre capitale pour débiter le nom du composant.

Voici à quoi va ressembler notre fichier `App.js` :

```
import './App.css';
import Personne from './Personne';

function App() {
  return (
    <div className="App">
      <Person />
    </div>
  );
}

export default App;
```

Tout commence avec la ligne suivante :

```
import Personne from './Personne.js';
```

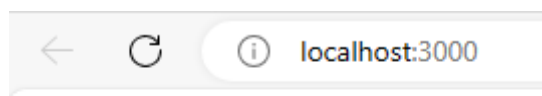
Ce qui aura plus de sens si vous êtes débutant parce que nous voulons accéder ou importer le composant `Personne` qui est créé dans le fichier `Personne.js`.

Mais `reactJs` nous permet d'omettre l'extension `.js` lors des importations.

```
import Personne from './Personne';
```

Notez que cette omission n'est autorisée que sur les fichiers JavaScript( `.js` ).

Comme résultat, nous voyons ceci dans notre navigateur :



**Je suis une personne**

Ce qui prouve que notre composant `Personne` est bel et bien créé. 😎

## Utilisation du code JavaScript dans un composant

Il arrive très souvent des situations dans lesquelles nous sommes obligé d'utiliser un code ou une variable JavaScript dans le composant que nous créons . Nous allons voir dans cette section comment le faire.

💡 **Application** : Dans notre composant `Personne` , nous allons changer l'ancien texte et mettre a la place le texte suivant : `Je suis une personne et j'ai x ans`

Dans ce texte, `x` représente un nombre aléatoire compris entre `0` et `50` .  
Nous pouvons obtenir `x` avec le code suivant :

```
Math.round(Math.random()*50);
```

Voici ce que le composant `Personne` va contenir pour le moment :

```
const Personne=()=>{  
  return(  
    <h2>Je suis une personne et j'ai Math.round(Math.random()*50) ans </h2>  
  )  
}  
export default Personne;
```

Comme résultat, nous avons ceci dans notre application :

**Je suis une personne et j'ai Math.round(Math.random()\*50)**

Ce qui veut dire que nous ne pouvons pas utiliser directement du code JavaScript dans le composant comme nous venons de le faire. 😞

**Alors quelle est la solution ?**

Eh bien, pour faire l'usage des variables ou du code JavaScript dans un composant comme dans le cas présent, nous devons englober ce code dans des accolades ( `{ }` )

Voici à présent le code du composant utilisant la variable JavaScript de façons correcte :

```
<h2>Je suis une personne et j'ai {Math.round(Math.random()*50)} ans </h2>
```

De cette manière, nous pouvons avoir un bon résultat comme sur le fichier ci-dessous :

```
C:\Windows\System32\cmd.exe
```

```
D:\projet>
```

Il est important de maîtriser et de bien comprendre ce que nous venons de faire ici parce que nous allons beaucoup l'utiliser dans la suite de ce chapitre.

D'ailleurs, vous l'utiliserez énormément dans vos futures applications `reactJs`.

## L'usage des attributs sur nos composants: props

En `HTML` normal, nous pouvons écrire ceci :

```
<input type="text" id="nom" />
```

Ce code crée dans la page web une zone de saisie de texte. Dans ce code, les éléments `type` et `id` sont appelés `attributs`.

Nous pouvons faire la même chose sur nos composants `reactJs`. Nous aurons ainsi la possibilité d'accéder aux valeurs des attributs que nous passerons aux composants au sein des fichiers JavaScript qui gèrent la création de ce composant en question.

En effet, nous pourrons utiliser le composant `Personne` comme suit :

```
<Personne nom="Alex" age="25" />
```

Et dans le fichier `Personne.js`, nous nous pourrons récupérer la valeur de l'attribut `age` pour rendre dynamique le composant `Personne`.

```
const Personne=(props)=>{
  return(
    <h2>Je suis une {props.nom} et j'ai {props.age} ans</h2>
  )
}
export default Personne;
```

Avec le code précédent, nous devons avoir à l'écran ce message :

**Je suis une Alex et j'ai 25 ans**



Nous remarquons, dans la fonction de création du composant `Personne`, un argument du nom de `props`. Cet argument représente un objet contenant l'ensemble des attributs passé à notre composant `Personne` (`nom` et `age`).

Étant donné que `props` est un objet, nous pouvons accéder aux valeurs des attributs du composant `Personne` en faisant ceci :

```
props.nom // récupération du nom
props.age // récupération de l'âge
```

NB: Connaissant les attributs qui composent l'objet, nous pouvons aussi décomposer ou déstructurer l'objet `props` au sein de la fonction `Personne` comme suit :

```
const Personne=({nom,age})=>{
  return(
    <h2>Je suis une {nom} et j'ai {age} ans</h2>
  )
}
export default Personne;
```

Ce qui fait que nous n'avons plus besoin d'utiliser `props`. Ainsi, on peut accéder aux attributs directement avec leur nom sans passer par `props`.

**Regardez le code HTML suivant :**

```
<button>Valider</button>
```

Ce code crée un bouton. Notez toujours que nous pouvons aussi utiliser le composant `Personne` de la même manière en ajoutant du texte à l'intérieur :

```
<Personne nom="Alex" age="25">J'aime l'informatique </Personne>
```

Tout comme les attributs `nom` et `age`, nous pouvons accéder au contenu qui se situe à l'intérieur de la balise `Personne` grâce à un attribut spécial nommé `children`.

Regardez plutôt :

```
const Personne=({nom,age,children})=>{
  return(
    <div>
      <h2>Je suis une {nom} et j'ai {age} ans</h2>
      <p>{children}</p>
    </div>
  )
}
```

```
    </div>
  )
}
export default Personne;
```

Le composant `Personne` peut donc être réutilisé plusieurs fois à des endroits différents dans notre application.

Voici un exemple d'utilisant :

```
import './App.css';
import Personne from './Personne';

function App() {
  return (
    <div className="App">
      <Personne nom="Alex" age="25" >J'aime la programmation</Personne>
      <Personne nom="Bob" age="28" >J'aime la musique</Personne>
    </div>
  );
}

export default App;
```

Nous aurons dans le navigateur le résultat suivant :

**Je suis une Alex et j'ai 25 ans**

l'aime la programmation

**Je suis une Bob et j'ai 28 ans**

l'aime la musique

## Gestion des states

Contrairement au `props`, les `states` jouent un rôle beaucoup plus avancé pour la gestion du composant dans lequel ils appartiennent. Tout changement de la valeur d'un `state` provoque immédiatement le réaffichage (re-rendering) du composant dans lequel il est créé.

## Dans quel cas pouvons-nous utiliser les states ?

La plupart du temps, cela vous paraîtra évident si un composant a besoin d'un `state` ou pas. Vous n'êtes même pas obligé de le savoir au début de la création du composant. Au fur et à mesure que votre application grandie, vous pouvez revenir et créer des `states` dans vos composants si le besoin se fait sentir.

Mais pour répondre à la question, nous pouvons dire que si un composant renferme un élément qui doit subir des modifications, alors, il faut utiliser un `state` pour gérer cet élément.

### Exemple

Supposons que notre composant `Personne` à un élément photo de profil. L'utilisateur peut changer sa photo de profil à tout moment. C'est donc logique dans ce cas d'utiliser un `state` pour gérer cela à l'intérieur du composant `Personne`.

La différence majeure qui existe entre les `states` et les `props` est que les `states` sont créés à l'intérieur du composant alors que les `props` sont créés à l'extérieur du composant.

## Création des states

Pour créer un state, nous allons utiliser un `Hook` spécial appelé `useState`. Nous parlerons plus tard de hooks et leur importance en `ReactJS`.

Nous devons déjà faire appel ou importer le `Hook useState` dans notre composant de cette façon:

```
import {useState} from "react";
```

Cette importation nous donne accès à la création et à la modification de `states` au sein du composant.

Mettons en place dans le composant `Personne` un `state` pouvant gérer la photo de profil :

```
const [photo,set_photo]=useState(null);
```

Nous venons juste de mettre en place le `state` avec la ligne ci-dessus.

L'usage de `useState` est un peu spécial. Selon la ligne du code ci-dessus, le `Hook useState` renvoie un tableau contenant deux éléments :

- `photo` : ce premier élément du tableau représente la `valeur initiale` du `state`. Dans notre cas, il va contenir le nom de la photo de la personne. Nous avons mis sa valeur initiale à `null`
- `set_photo` : représente la fonction qui va se charger de modifier la valeur de la photo. Étant donné que `set_photo` est une fonction, nous pouvons donc l'appeler à tout moment en

faisant ceci :

```
set_photo("nouvelle photo");
```

Voici le code complet de l'état actuel de notre composant `Personne` :

```
import {useState} from "react";
const Personne=({nom,age,children})=>{
  const [photo,set_photo]=useState(null);

  return(
    <div>
      <h2>Je suis une {nom} et j'ai {age} ans</h2>
      <p>{children}</p>
    </div>
  )
}
export default Personne;
```

### Exercice :

Mettons en place dans le composant `Personne`, un `state` pouvant nous permettre de changer/modifier le nom de la personne.

*NB : le composant doit alors disposer d'une zone de texte pour permettre la modification du nom.*

*La valeur initiale du `state` doit être le nom passé en attribut.*

### Solution :

```
import {useState} from "react";

const Personne=({nom,age,children})=>{
  const [name,set_name]=useState(nom);

  return(
    <div>
      <input type="text" value={name} onChange={(e)=>set_name(e.target.value)}/>
      <h2>Je suis une {name} et j'ai {age} ans</h2>
      <p>{children}</p>
    </div>
  )
}
```

```
}  
export default Personne;
```

Nous avons mis l'attribut `onChange` sur la zone de texte qui appelle la fonction de modification du nom qui est `set_name`.

Comme vous pouvez le voir, le composant est réaffiché à chaque fois que nous appelons `set_name`.

Ce qui explique encore mieux le fait que: *si le state d'un composant est modifié, cela provoque une actualisation du composant concerné.*

## Application d'un style CSS

Dans une application web, il est presque impossible de ne pas faire appel à `CSS` pour designer nos composants.

Il faut noter que nous utilisons `ReactJS` pour le front-end c'est-à-dire le côté client (visuel) de notre application, il est donc essentiel d'utiliser du `CSS` pour formater nos composants. C'est donc le but de cette section.

Le composant créé dans le fichier `Personne.js` se présente actuellement comme ceci :

**Je suis une Alex et j'ai 25 ans**

J'aime la programmation

**Je suis une Bob et j'ai 28 ans**

J'aime la musique

Si nous voulons par exemple appliquer une bordure noire à ce composant, il faut bien évidemment lui appliquer un style.

Pour cela, nous allons attribuer au composant `Personne` une classe avec l'attribut `className`.

Nous n'avons pas utilisé `class` comme attribut parce que cela représente un mot clé réservé en JavaScript.

Le nom `className` est alors utilisé à la place de `class` qui est utilisé en pur `HTML` sans `ReactJS`. Mais une fois dans le navigateur, on verra `class` au lieu de `className`.

```
<div className="personne">
  <input type="text" value={name} onChange={(e)=>set_name(e.target.value)}/>
  <h2>Je suis une {name} et j'ai {age} ans</h2>
  <p>{children}</p>
</div>
```

Une fois la classe définie, il faut alors définir les styles CSS de la classe `personne`. Nous allons pour se faire créer un fichier nommé `Personne.css` dans le dossier `src` et nous allons ajouter les codes CSS pour formater notre composant `Personne`.

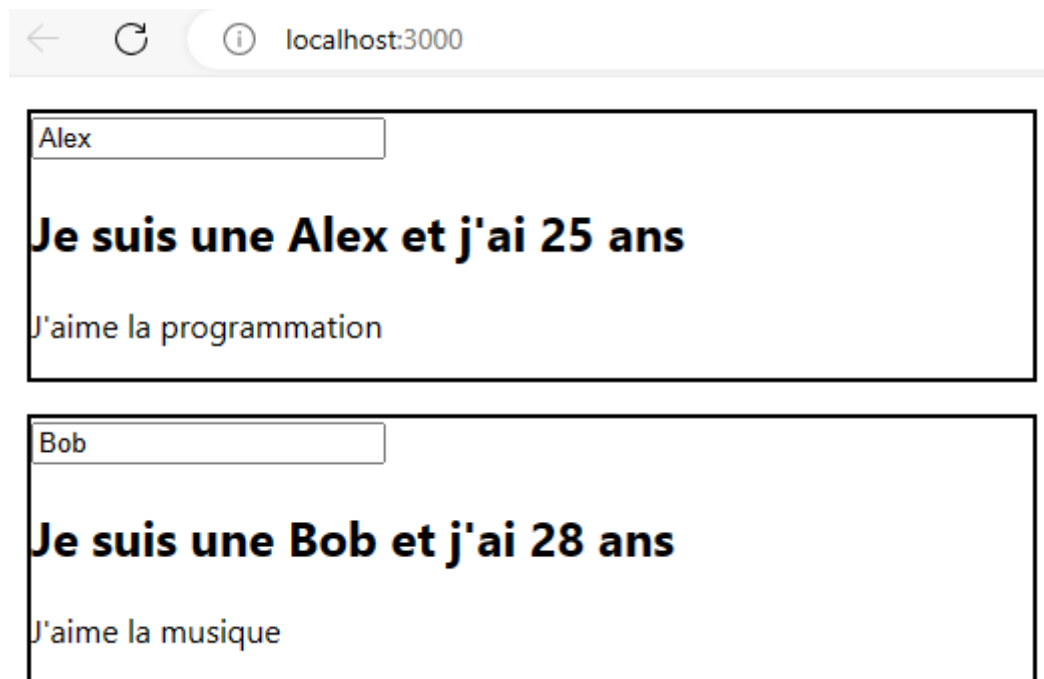
Voici le contenu de ce fichier `Personne.css`

```
.personne{
  border:2px solid black;
  width:500px;
  margin:1rem;
}
```

Le fichier `Personne.js` doit faire appel à ce fichier `Personne.css` :

```
import "../Personne.css";
```

Maintenant regardons dans le navigateur si nous avons un changement :



Voici le code complet du fichier `Personne.js`

```
import {useState} from "react";
import "./Personne.css";

const Personne=({nom,age,children})=>{
  const [name,set_name]=useState(nom);

  return(
    <div className="personne">
      <input type="text" value={name} onChange={(e)=>set_name(e.target.value)}/>
      <h2>Je suis une {name} et j'ai {age} ans</h2>
      <p>{children}</p>
    </div>
  )
}
export default Personne;
```

Ce qui montre que notre code `CSS` a bien marché sur notre composant `Personne`. Nous pouvons à partir de maintenant écrire toute sorte de code `CSS` pour formater nos composants.

NB : pour des raisons de simplicité, il est préférable que pour chaque composant que vous créez dans un fichier JavaScript, créez aussi un fichier `CSS` du même nom dans lequel vous mettrez tous les codes liés au design de ce composant.

Si vous maîtrisez un Framework CSS comme `Tailwindcss` ou `Bootstrap`, nous vous conseillons de l'utiliser au lieu du pur CSS.

## Hooks

Cette partie concernant les hooks est une nouveauté de `ReactJS` qui demande un peu de patience pour une bonne maîtrise si vous êtes un débutant.

Les hooks sont en quelque sorte des fonctions qui vont nous aider à se brancher sur des composants `reactJS` pour acquérir des fonctionnalités supplémentaires.

Nous allons ainsi parcourir quelques hooks les plus utilisés dans une application `ReactJS`.

### useState

Ce hook nous permet de gérer les états locaux d'un composant. Nous l'avons déjà utilisé au niveau du composant `Personne` pour gérer le changement du nom de l'utilisateur.

En effet, le nom de l'utilisateur est un élément qui se trouve à l'intérieur du composant. Et pour pouvoir agir (modifier) sur cet élément, il faut utiliser le hook `useState`.

## Usage

`useState` est une fonction tout comme les autres hooks que nous allons parcourir par la suite. Et cette fonction prend en paramètre une valeur initiale du state et nous renvoie ensuite un tableau contenant deux valeurs. La première valeur étant l'état actuel de l'élément et la seconde représente la fonction qui va modifier l'état.

Avec ce code ci-dessus :

```
const [x, set_x]=useState(0);
```

- `x` représente l'état que nous voulons gérer qui a pour valeur `0` mise dans la parenthèse de `useState`
- `set_x` est la fonction qui modifie la valeur `x`

```
set_x(3);
```

Avec ce code, la valeur de `x` devient `3` et le composant est réaffiché automatiquement selon la nouvelle valeur du state `x`.

## useEffect

Supposons que nous voulons exécuter un code juste après l'affichage ou la mise à jour du DOM. Dans ce cas, il est fort probable que vous aurez besoin d'utiliser le hook `useEffect`.

Voici deux exemples de cas dans lesquelles nous pouvons songer à utiliser ce hook

`useEffect` :

- Lancer une requête réseau après changement de la page web,
- modification manuelle du DOM.

## Usage

`useEffect` est une fonction tout comme `useState` que vous avez déjà vue. Il prend en paramètre une autre fonction qui contient le code à exécuter après réaffichage du DOM.

```
useEffect(()=>{  
  console.log("Salut")  
})
```

La fonction contenue dans le `useEffect` sera appelée toutes les fois que le DOM est réaffiché ou qu'un `state` quelconque du composant est modifié.



Tout comme `useState` , nous devons importer le `useEffect` avec la ligne ci-dessous avant de pouvoir l'utiliser au sein d'un composant :

```
import {useEffect} from "react";
```

Ajoutons à notre composant `Personne` un autre `state` permettant de modifier L'âge:

```
const [m_age, set_m_age]=useState(age);
```

Nous avons maintenant deux states ( `name` et `m_age` ) dans notre composant.

Ajoutons le code de `useEffect` vue précédemment à ce composant. Le code complet du composant `Personne` doit être ceci :

```
import {useState,useEffect} from "react";
import "./Personne.css";

const Personne=({nom,age,children})=>{
  const [name,set_name]=useState(nom);
  const [m_age,set_m_age]=useState(age);

  useEffect(()=>{
    console.log("Salut")
  })

  return(
    <div className="personne">
      <input type="text" value={name} onChange={(e)=>set_name(e.target.value)} />
      <input type="text" value={m_age} onChange={(e)=>set_m_age(e.target.value)} />
      <h2>Je suis une {name} et j'ai {age} ans</h2>
      <p>{children}</p>
    </div>
  )
}
export default Personne;
```

Dans la console du navigateur, nous devons déjà voir le texte `Salut` s'afficher après chargement de la page.

Essayons de saisir dans la zone de texte pour modifier le nom de la personne. Nous verrons aussi que la fonction à l'intérieur du hook `useEffect` est encore exécutée et nous affiche `Salut` dans la console.

La même chose se répète si nous essayons de saisir dans la zone de texte pour la modification de l'âge de la personne.

Vous comprenez désormais que lorsque nous saisissons dans la zone du texte pour le nom ou l'âge cela appelle respectivement `set_name` et `set_m_age` provoquant ainsi l'actualisation du DOM.

Et une fois le DOM modifié, notre `useEffect` exécute le code dans la fonction qui lui est passé en paramètre.

Notez que nous pouvons utiliser plusieurs `useEffect` dans le même composant pour effectuer des tâches spécifiques.

Ces `useEffect` seront exécutés selon l'ordre dans lequel nous les avons mis dans le composant.

Nous pouvons aussi limiter le moment d'exécution de la fonction se trouvant à l'intérieur du `useEffect`.

Supposons que nous voulons que la fonction `useEffect` soit exécuter seulement quand le `state` du `nom` change.

Dans ce cas, il faut passer comme second paramètre à la fonction `useEffect` un tableau avec pour valeur le `state` `nom` en faisant ceci :

```
useEffect(()=>{
  console.log("Salut")
},[name])
```

Après exécution, nous allons voir que la fonction du `useEffect` n'est exécutée qu'au chargement du DOM et aussi en saisissant dans la zone de texte du nom. Rien ne se passe si nous écrivons dans la zone de texte de l'âge.

Nous pouvons ajouter d'autres states dans le second argument du `useEffect` selon nos besoins.

## Autres hooks

Nous venons de voir les deux hooks les plus utilisés de `reactJS`. Toute fois, nous avons d'autres hooks qui sont aussi très utilisés. Une bonne maîtrise de `useState` et `useEffect` vous permettra de comprendre et d'utiliser facilement les autres hooks comme suit :

- `useRef`
- `useLayoutEffect`
- `useCallback`

- useMemo
- useReducer
- useContext

## Gestion avancée des states

Dans un gros projet, l'usage de `state` ne suffira pas pour affronter toutes les exigences d'un projet web. C'est pour cela que nous avons une librairie appelée `Redux` qui peut nous aider à mieux gérer les `states`. Faites des recherches personnelles sur `Redux` pour en savoir plus. Mais vous pouvez très bien faire un projet sans utiliser `Redux`.

Référence: <https://redux.com>

## ROUTER

Nous ne pouvons mettre fin à ce chapitre concernant `ReactJS` sans parler du concept de `Router` de `reactJS` qui est une fonctionnalité nous permettant de créer des liens entre les pages.

Le module `react-router-dom` peut vraiment nous être utile pour réaliser cette tâche. Vous pouvez l'installer avec la ligne suivante :

```
npm install react-router-dom
```

Allez sur le site officiel de ce module pour en savoir davantage.

Référence: <https://react-router-dom.com>

# Annexes

# Live-server

`Live-server` un outil qui vous permet d'exécuter une application web comme si elle était sur un vrai serveur web.

Supposons que nous avons un dossier `projet` sur notre disc local `D` et qu'à l'intérieur de ce dossier, nous avons une page web `index.html`.

Si nous ouvrons ce fichier en double-cliquant dessus, nous aurons dans la barre d'adresse du navigateur l'adresse suivante:

`file:///D:/projet/index.html`

Cela ne pose pas de problème en tant que tel, mais certaines fonctionnalités ou techniques lors du développement web exigent une réelle adresse web ou du moins la simulation d'une vraie adresse web. C'est le cas de la fonction `fetch` de JavaScript qui ne peut pas fonctionner avec cette façon d'ouvrir une page web.

`Live-server` est l'un des outils que nous pouvons utiliser pour nous aider à ouvrir une page web pour qu'elle soit compatible avec toutes les fonctionnalités et techniques liées aux développements des applications web.

## Installation de live-server

L'installation de `live-server` est très facile, il faut ouvrir l'invite de commande sur votre ordinateur et saisir la commande suivante :


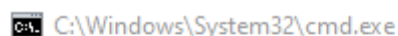
```
npm install -g live-server
```

La présence de `npm` signifie que vous devez préalablement installer `nodeJS` sur votre ordinateur. Lisez le chapitre concernant l'installation de `nodeJS` si vous ne l'avez pas sur votre ordinateur.

## Lancer un projet web avec live-server

Prenons toujours le cas de notre dossier `projet` qui se trouve sur le disc local `D` et qui contient la page web `index.html`.

Pour ouvrir ce projet avec `live-server`, nous devons aller dans l'invite de commande en allant sur le disc local `D` et ensuite rentrer dans le dossier `projet`.



Une fois dans le dossier de notre projet, il suffit de saisir la commande suivante :

```
live-server
```

Le projet sera automatiquement lancé dans un navigateur sur l'adresse <http://127.0.0.1:8080/> ou sur <http://localhost:8080>

*NB : Un autre avantage à retenir à propos de `live-server` est que, lorsque vous écrivez du code dans votre éditeur de texte, la page web lancée avec `live-server` est automatiquement actualisée. Vous n'avez plus besoin de faire l'actualisation manuelle dans le navigateur pour voir vos modifications. Cela peut vous aider à aller beaucoup plus rapidement dans le développement de vos applications.*

## Installation de nodeJS

`NodeJS` est un langage de programmation coté serveur, mais il est avant tout un logiciel ou un outil que vous utiliserez très souvent dans le développement web même si vous n'allez pas forcément écrire du code `nodeJS`.

L'installation de plein d'outils dans le monde de la programmation web requiert la présence de `nodeJS` sur votre ordinateur. Vous devez donc forcément l'installer sur votre ordinateur si vous êtes développeur web.

Allez sur le lien de téléchargement sur l'adresse <https://nodejs.org/en/download/> et téléchargez la version de nodeJS qui correspond à votre système d'exploitation.

Après le téléchargement, il faut l'installer sur votre ordinateur comme tout autre logiciel normal.

Une fois terminé avec l'installation, ouvrez un nouvel invite de commande et saisissez la commande suivante :

```
node --version
```

Vous devez voir le numéro de version du `nodeJS` que vous venez d'installer sur votre ordinateur.

Si vous avez une erreur, alors, `nodeJS` n'est pas bien installé, ou une erreur est survenue lors de l'installation de `nodeJS`.