Apprentissage des langages Web

Réalisé par : SERRES Arnaud
L1-MPCI

TUTEUR DE STAGE : BRUCKER François

Juin 2024

Table des matières

1 Introduction

1.1 Objectifs

L'objectif de mon stage était de mettre en place une plateforme d'apprentissage des langages web.
En m'appuyant sur le cours de Mr.Brucker, j'ai tout d'abord dû apprendre les différents aspects des langages web, puis une fois que je maitrisais suffisement le sujet, je devais mettre en place le plan d'un cours efficace, et établir différents supports, tels que des pages web, afin de rendre l'apprentissage plus facile.

1.2 Le LIS

Le Laboratoire d'Informatique et Systèmes (LIS) est un centre de recherche d'informatique situé à Aix-Marseille. Le LIS est affilié à Aix-Marseille Université, l'Université de Toulon, au CNRS, et à l'École Centrale de Marseille. Il regroupe environ 350 chercheurs, enseignants-chercheurs, et doctorants.

Le LIS travaille dans quatre domaines de recherche : les calculs, la science des données, l'analyse et le contrôle des systèmes et dans les signaux et images. Ces quatre domaines sont divisés en plusieurs sujets d'étude, dont les intelligences artificielles, les bases de données, l'aide à la décision et l'imagerie médicale.

Les recherches menées au LIS ont une finalisation dans divers domaines tels que le transport, la santé, l'environnement, l'énergie, la défense, etc.

2 Le front

Le front-end est une partie essentielle à la création de sites applications web, c'est la partie visuelle, que l'usager va voir et avec laquelle il va interagir. Les deux piliers du front sont le HTML (HyperText Markup Language) et le CSS (Cascading Style Sheets).
Le HTML est un langage de balisage utilisé pour structurer le contenu de notre page. Il permet de définir la hiérarchie des différents éléments présents sur une page web, tels que les titres, les différents paragraphes, les images, etc. Cette organisation est essentielle afin d'avoir une page intéractive.
Le CSS, quant à lui, est utilisé pour changer les couleurs, les polices, les marges, et permet d'animer les différents éléments HTML de la page.

2.1 HTML

Maîtriser l'HTML et le CSS est essentiel pour avoir des pages web claires et interactives. Il n'est malheureusement pas toujours facile de trouver une plateforme où apprendre facilement ces langages. Ayant dû tout apprendre depuis rien moi même, j'ai retracé mes difficultés et mis en place des pages web afin de concrétiser mes connaissances. Cela m'a permis d'établir un plan d'apprentissage.
Premièrement, il faut commencer à apprendre l'HTML, en effet celui-ci est essentiel pour pouvoir utiliser le CSS.
Comme dans le cours de Mr.Brucker, un bon moyen de commencer est d'aller sur une page web existante, puis, grâce aux outils de développement, de regarder le code HTML de la page et de voir comment chaque balise s'indente dans l'autre.
Après avoir vu de manière générale le fonctionnement de HTML, il faut voir séparément le fonctionnement de chaque balise importante, en regardant des exemples d'implémentations simples.
Le premier blocage que j'ai rencontré a été de lier les différentes balises entre elles. Afin de remédier à cela, mettre en place une page web comportant les différents éléments HTML que je venais de rencontrer m'a beaucoup aidé. Un exemple de projet pour finaliser la partie "Apprentissage des balises" pourrait-être :

> *Faire une page web sur son animal préféré, avec une image de cet animal et un lien vers sa page wikipédia.*

De plus, cette page peut servir de base pour faire un projet CSS plus tard.

2.2 CSS

Tout comme le HTML, le CSS est assez simple à apprendre. Le CSS consiste à faire des listes de style pour les éléments HTML de la page. Le meilleur moyen de commencer en CSS, c'est tout simplement d'apprendre les différentes options de personnalisation. Afin de créer une suite logique avec la partie HTML, on peut améliorer le style du document HTML qu'on a créé dans la première partie. Ce projet va nous suivre tout au long de notre apprentissage, et va permettre de mieux comprendre le lien entre les différentes parties de l'apprentissage.

Après avoir maîtrisé les balises HTML, il faut apprendre à positionner ces éléments sur la page. Les quatres positions principales sont “static”, “relative”, “absolue” et “fixed”. Voir visuellement les nuances parmi ces options est essentiel pour apprendre. C’est pour cela que j’ai réalisé trois pages web très visuelles qui permettent cela, il est primordial de regarder le code de ces pages web pour comprendre qu’est-ce qui fait quoi :

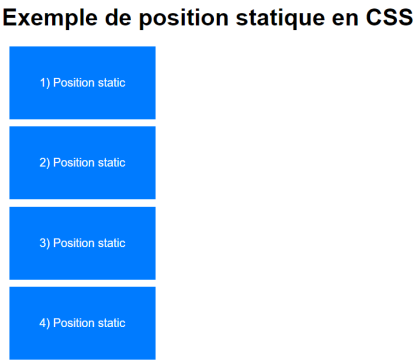

FIGURE 1 – Position static

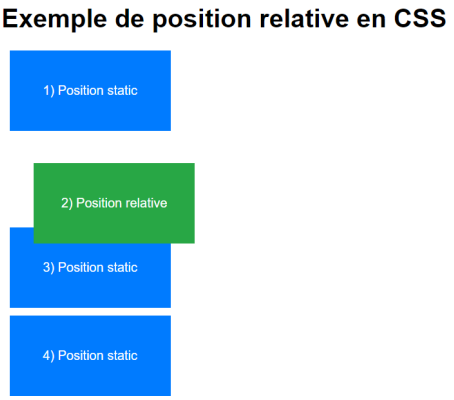

FIGURE 2 – Position Relative

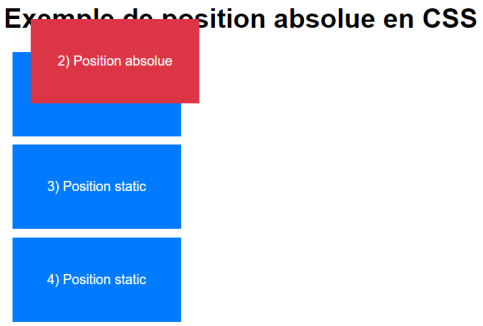

FIGURE 3 – Position Absolue

J’ai également créé une page pour la postion fixe, mais afin de voir son fonctionnement, il faut baisser la page (pour voir que l’element fixe ne bouge pas), et donc ce n’est pas pertinent d’en faire une capture d’écran.
Et celles-ci pour voir le fait que la position est “absolue” par rapport au premier parent placé :

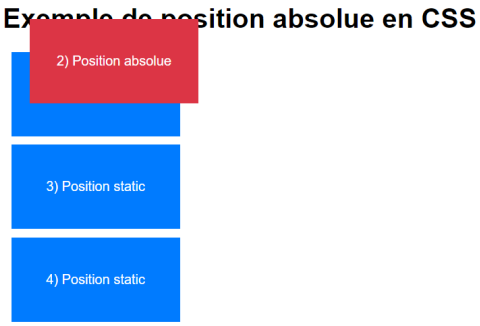

FIGURE 4 – Position absolue sans parent placé

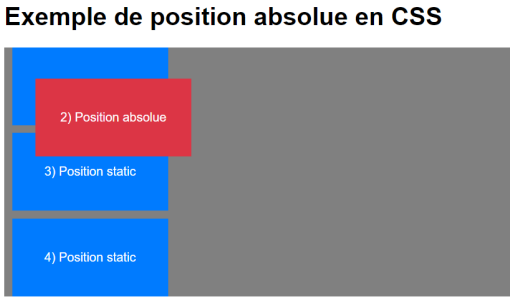

FIGURE 5 – Position absolue avec parent placé

D'autres aspects techniques peuvent-être explicités grâce à des pages web, par exemple "comment gérer l'overflow" :

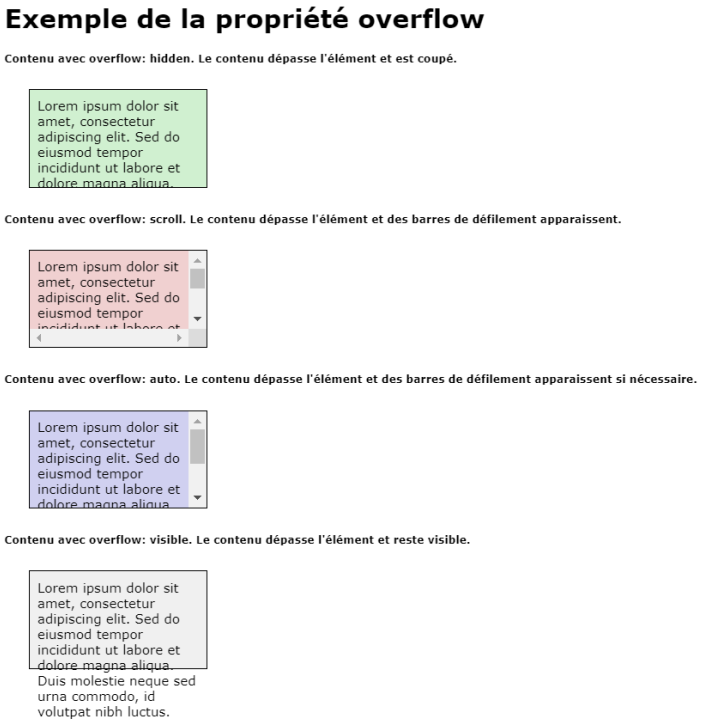

FIGURE 6 – Comment gérer l'overflow

Une manière efficace que j'ai trouvé pour m'améliorer en CSS, c'est de trouver un élément d'une page web à imiter (voir même toute la page web), et essayer d'en faire une copie exacte :

FIGURE 7 – Bouton classique

Par exemple, pour imiter ce bouton classique des sites web, j'ai appris beaucoup de notions CSS comme comment faire des coins ronds, la propriété :hover d'un élément HTML, qui permet de détecter quand la souris passe au-dessus de cet élément, et bien d'autres. En répétant ce procédé quelques fois, on acquiert une dextérité avec le CSS assez rapidement.

2.3 Les displays

Grid et Flexbox sont deux outils CSS indispensables afin de bien positionner les éléments sur l'écran. Ils permettent à la page web d'être adaptée à toutes tailles d'écrans, et proposent des fonctionnalités très utiles pour un développeur front.
C'est également un des plus gros obstacles que j'ai rencontré lors de mon stage. Flexbox en particulier est très peu naturel à utiliser. L'erreur que j'ai faite lors de mon apprentissage de grid et flexbox a été de m'appuyer sur des cours écrits au lieu de vidéos. En effet sur les cours écrits il n'y avait pas d'animations, on ne voyait que le résultat, et pas le chemin pour arriver à ce résultat etc.
J'ai donc pris les avantages des vidéos et j'en ai fait des pages web. Je compare les différentes commandes avec un système de boîtes, qui permet de comprendre de manière claire ce que chaque commande fait.
Pour bien comprendre ce qu'il se passe, il faut regarder à la fois le code CSS et la page web.

Pour display :flexbox, j'ai explicité les commandes suivantes :

— **Container :**

- `flex-direction` : row, column ==> determine la direction dans laquelle les éléments vont être arrangés.
- `justify-content` : flex-end, center, space-between, space-around, space-evenly ==> arrange les éléments sur la direction de flex-direction
- `align-items` : flex-end, center, baseline ==> arrange les éléments selon la direction inverse à celle de flex-direction
- `flex-wrap` : wrap ==> retourne à la ligne lorsqu'il manque de place.
- `gap` : (x, y) ==> espace entre deux éléments selon l'horizontale et la verticale

— **Item :**

- `flex-grow` : 1 : grandit s'il peut grandir
- `flex-shrink` : coefficient de rétrécissement quand la page est rétrécie : Par exemple `flex-shrink: 5` l'élément diminue 5 fois plus vite que les autres.

- `align-self` : comme `align-items`, mais pour un seul élément
- `order` : ex : $1 > 0$ donc l'élément dont l'ordre est supérieur se rangera en fin de ligne/colonne

Voici quelques unes des pages web servant à illustrer ces commandes :

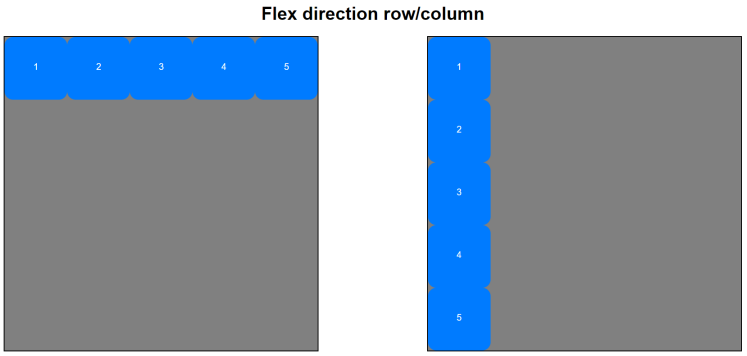

FIGURE 8 – flex-direction

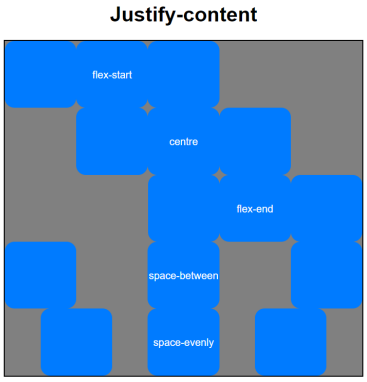

FIGURE 9 – Justify content

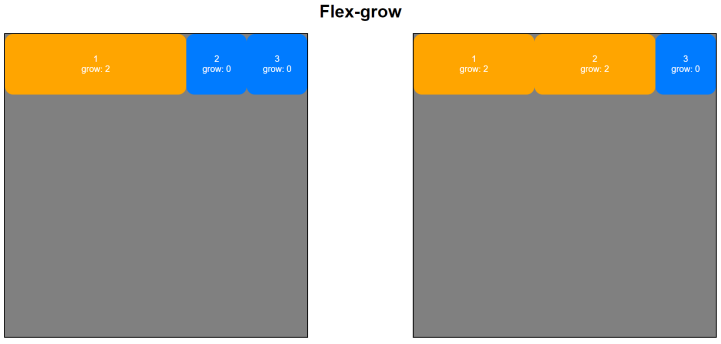

FIGURE 10 – Grow

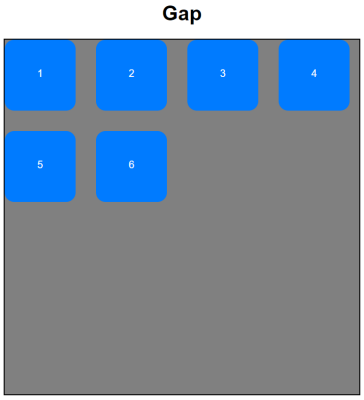

FIGURE 11 – Gap

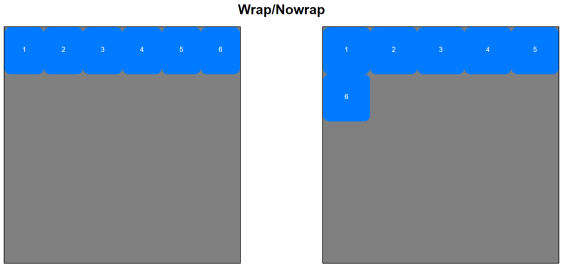

FIGURE 12 – Wrap

Il faut bien regarder le code de chaques pages en complément du résultat visuel. De même, j'ai mis en place des pages web pour les commandes suivantes :

- **Container :**
  - `grid-template-rows` : ex. 100px 100px pour deux lignes de 100px de hauteur.
  - `grid-template-columns` : ex. 1fr 1fr pour deux colonnes de hauteurs égales.
  - `grid-gap` : (espacement rows, espacement columns)
  - `justify-items` / `align-items` : prend les mêmes paramètres que pour `flexbox`, mais le paramètre de base est `stretch`.
- **Item :**
  - `grid-row/column-start` : là où commence la ligne/colonne (inclus)
  - `grid-row/column-end` : là où finit la ligne/colonne (exclus)
  - `grid-area` : ligne début/colonne début/ligne fin/colonne fin ==> remplace les deux lignes ci-dessus

Et voici quelques unes des pages web représentant les commandes :

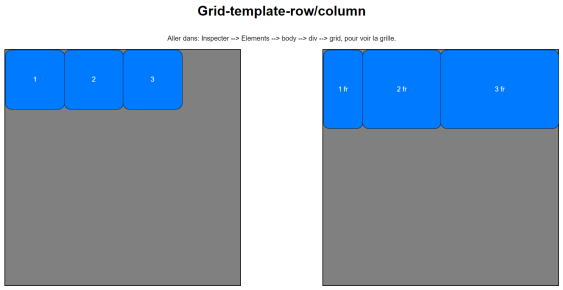

FIGURE 13 – Grid-template

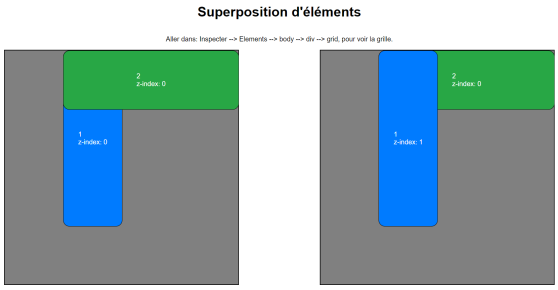

FIGURE 14 – Index

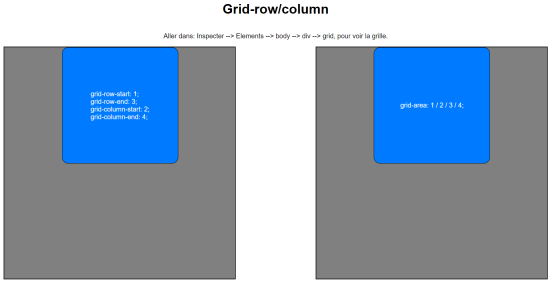

FIGURE 15 – Grid-row/column

2.4 Les bibliothéques

De multiples bibliothèques CSS sont disponibles gratuitement, leur utilisation est très répandue car elle permet aux développeurs front-end de gagner du temps et d'avoir une interface bien plus attrayante. En effet, ces dernières fournissent aux utilisateurs des modèles pré-fabriqués.

J'ai seulement étudié la bibliothèque bootstrap car c'est la bibliothèque en vogue en 2024, et celle qu'un apprentis développeur front pourrait utiliser. Elle permet entre autres de créer une page dite "responsive" : qui s'adapte à tout format d'écran. Après avoir fait du CSS, qui est un langage vraiment naturel, j'ai eu un peu de mal sur quelques aspects de Bootstrap, en particulier le système de grille, les "break-points" et les boutons. Ce qui m'a vraiment débloqué, c'est de voir visuellement ce que chaque commande représentait, j'ai donc fait quelques pages web. Voici une page web permettant de comparer la grille de bootstrap (à droite) à un positionnement fait grâce à flexbox (à gauche), et le code associé à la grille de bootstrap :

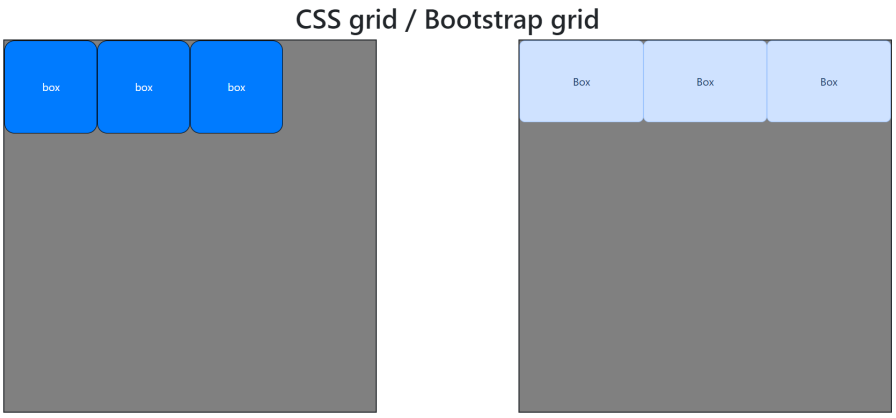

FIGURE 16 – Comparaison de la grille de bootstrap (à droite) à un positionnement flexbox (à gauche)

```
<div class="row">
    <div class="col p-5">
        Box
    </div>
    <div class="col p-5">
        Box
    </div>
    <div class="col p-5">
        Box
    </div>
</div>
```

FIGURE 17 – Code de la grille Bootstrap

Les “breakpoints” servent à différencier les différentes tailles d’écran, les différents formats sont : None ==> sm ==> md ==> lg ==> xl ==> xxl où la taille None représente le format telephone (Bootstrap est “mobile first” donc ses paramètres par défaut sont fait pour les téléphones). J’ai trouvé cela très complexe à comprendre, en effet on a pas l’habitude de prendre en compte la taille d’écran de l’utilisateur. Ce qui m’a débloqué, c’est d’avoir fait des pages web, et d’en changer la taille grâce aux outils de développeur. Ainsi j’ai pû comprendre toutes les subtilités des “breakpoints”. Cette page explique à la fois aussi bien les propriétés de "l’offset" que des breakpoints, "offset-md-4 offset-lg-0" permet de préciser que l’offset est nul jusqu’à ce que la taille de l’écran devienne moyenne, où l’offset est de 4, puis il est de nouveau nul à partir d’un écran large :

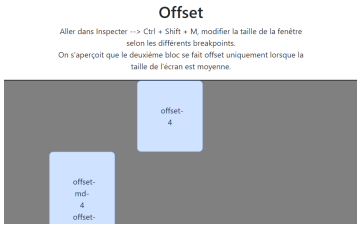

FIGURE 18 – Petite taille d’écran

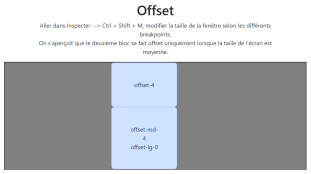

FIGURE 19 – Taille d’écran moyenne

FIGURE 20 – Grande taille d’écran

Pour illustrer le temps gagné à utiliser une bibliothèque par rapport à faire le style de la page en CSS, j’ai reproduit une carte Bootstrap (qui est créée automatiquement grâce à son modèle prévu), en CSS. La carte en Bootstrap m’a pris 5 minutes à faire alors que la carte en CSS m’a pris 45 minutes à faire, et l’esthétique n’est pas aussi agréable que celle en Bootstrap :

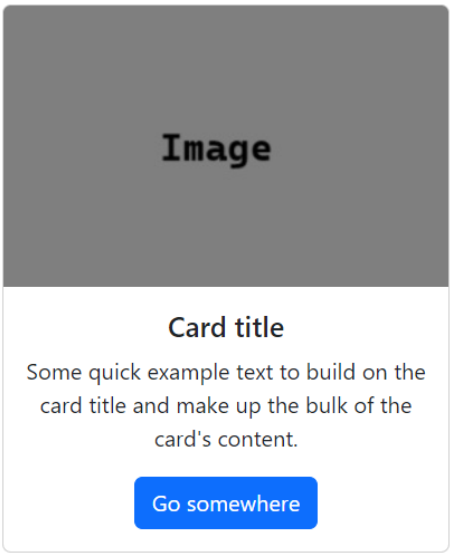

FIGURE 21 – Carte Bootstrap

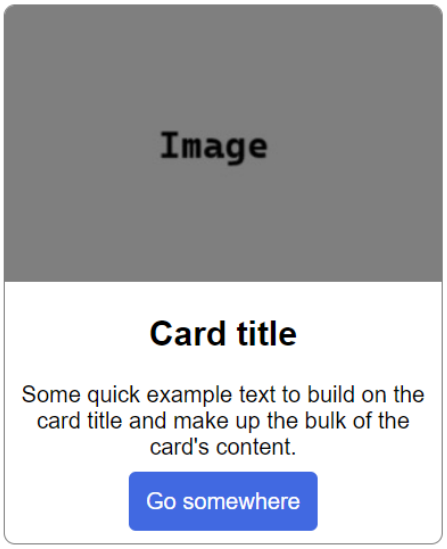

FIGURE 22 – Carte CSS

Le meilleur moyen d'apprendre une bibliothèque, c'est de lire la doc, et de l'appliquer en cas concret. On peut par exemple essayer d'imiter en bootstrap la page HTML et CSS qu'on a fait évoluer jusqu'à là :

FIGURE 23 – Page en Bootstrap

FIGURE 24 – Page en CSS

3 Le JavaScript

3.1 Javascript et l'arbre DOM

JavaScript est un langage de programmation interprété. Il est utilisé à la fois pour le front-end et le back-end. Son utilisation dans le front permet de rendre les pages web interactives en détectant les actions des utilisateurs, telles que les clics, les mouvements de souris, ou bien encore les boutons appuyés.

Le DOM représente la page web sous la forme d'une arborescence d'objets HTML.

Le javascript peut directement interférer avec l'arbre DOM en ajoutant des éléments HTML ou en en supprimant en temps réel. Cela a un grand potentiel pour rendre les pages web interactives par exemple en affichant des messages ou en enclenchant des animations avec un clic.

Pour utiliser JavaScript afin de manipuler le DOM, il faut d'abord comprendre comment sélectionner les éléments HTML et comment les modifier.

Le cours de Mr.Brucker explique le fonctionnement de nombreuses commandes en même temps grâce à **un** gros projet. C'est très utile de voir comment tout s'agence ensemble, mais j'ai eu du mal à tout incorporer d'un coup. J'ai donc divisé ce gros projet en trois plus petits projets. Chacun de ces petits projets présente une commande principale, mais on y utilise également d'autres commandes. L'avantage principal à procéder ainsi est que chaque projet est assez simple pour pouvoir comprendre le fonctionnement de chaque commande, contrairement à un gros projet, où on pourrait confondre l'effet d'une commande avec celui d'une autre.

Ainsi, j'ai explicité les commandes createElement, appendChild, getAttribute et setAttribute. C'est assez difficile de trouver une manière interactive pour expliquer cela, il faut pouvoir changer l'arbre DOM même après avoir chargé la page. J'ai opté pour exécuter la commande au clic d'un bouton. Pour les commandes createElement et appendChild, ça donne :

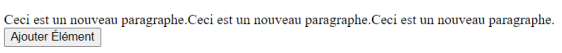

FIGURE 25 – Page appendChild

```
<!DOCTYPE html>
<html>
<head>
    <title>Utilisation de appendChild</title>
</head>
<body>
    <div id="conteneur"></div>
    <button onclick="appendElement()">Ajouter Élément</button>
    <script>
        function appendElement() {
            var nouveauParagraphe = document.createTextNode("Ceci est un nouveau paragraphe.");
            var conteneur = document.getElementById("conteneur");
            conteneur.appendChild(nouveauParagraphe);
        }
    </script>
</body>
</html>
```

FIGURE 26 – Code appendChild

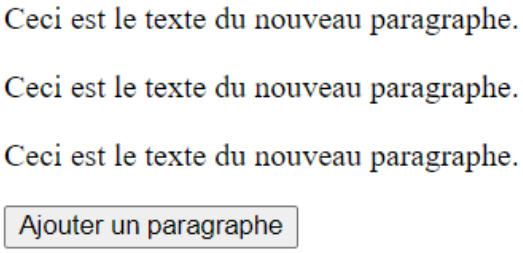

FIGURE 27 – Page createElement

```html
<!DOCTYPE html>
<html>
<head>
    <title>Utilisation de createElement et appendChild</title>
</head>
<body>
    <div id="conteneur"></div>
    <button onclick="ajout_paragraphe()">Ajouter un paragraphe</button>
    <script>
        function ajout_paragraphe() {
            var nouveauParagraphe = document.createElement("p");
            var texte = document.createTextNode("Ceci est le texte du nouveau paragraphe.");
            nouveauParagraphe.appendChild(texte);
            var conteneur = document.getElementById("conteneur");
            conteneur.appendChild(nouveauParagraphe);
        }
    </script>
</body>
</html>
```

FIGURE 28 – Code createElement

3.2 Communication avec un serveur

L'utilité de Javascript dans le front-end n'est pas seulement de pouvoir interagir avec l'arbre DOM, c'est aussi de faire le lien avec le back-end. Grâce à JavaScript, on peut envoyer et recevoir des données du serveur sans avoir à recharger la page. Cette capacité est essentielle pour avoir une page web rapide.

La commande Javascript utilisée pour communiquer avec le serveur est "fetch", elle peut être trouvée sous différentes formes :

— GET : Récupère des données du serveur.
— POST : Envoie des données au serveur.
— PUT : Met à jour les données sur le serveur.
— DELETE : Supprime des données du serveur.

La "commande" fetch est faite pour optimiser la performance de la page. Après avoir fait sa requête auprès du serveur, elle va passer à autre chose en attendant sa réponse.
La difficulté avec cette commande, réside dans la multitude de règles qu'il y a autour :

- Il faut s'assurer d'utiliser la bonne forme (GET, POST, PUT ou DELETE), celle que le serveur attend.
- Il faut que la nature de l'objet que l'on envoie soit bien celle attendue par le serveur (généralement un fichier json, mais il peut y avoir des exceptions).
- Il faut savoir ce que le serveur rend, sous quelle forme cela se présente, et adapter notre code en conséquence.
- Il y a beaucoup de sources d'erreurs possibles (par exemple le serveur peut être en maintenance), il faut donc gérer le cas où il y a une erreur.
- L'utilisation de cette commande requiert beaucoup de lignes de code, il est facile qu'une erreur se glisse dedans.

De plus, cette dernière reste assez incertaine. En effet, le temps de réponse dépend de beaucoup de paramètres dont la taille de la réponse. Toutes ces spécificités rendent l'utilisation de la méthode fetch très difficile.
Pour y voir plus clair, j'ai créé de multiples projets. Dans le premier, j'ai essayé d'importer un fichier léger (ne contenant qu'un mot) et un fichier très lourd (un livre en format numérique) simultanément, pour voir si la taille du fichier avait vraiment un rôle à jouer. Les résultats ont montré à quel point la méthode fetch est incertaine. Bien que dans la majorité des cas le fichier léger arrivait avant le fichier lourd, l'inverse peut arriver.
J'ai également essayé d'importer des fichiers depuis un serveur distant et un serveur local, afin de me familiariser avec toutes les utilisations du fetch (ce qui fut très utile pour mon projet final).

Je pense que la meilleure façon d'apprendre la commande "fetch" et le javascript en général, c'est la pratique. C'est les mêmes erreurs qui surgissent à chaque fois, et bien qu'elles soient très longues à comprendre et à résoudre les premières fois, au fur et à mesure qu'on pratique du javascript, on gagne en rapidité.

4 Communication Web et Gestion des Fichiers

4.1 URL

Les URL (Uniform Resource Locators) jouent un rôle essentiel dans la communication web, car elles définissent l'adresse des ressources sur le web. Comprendre comment elles sont construites et comment elles fonctionnent est primordial pour créer une page web évoluée. Bien que je n'avais jamais vraiment étudié les URLs, j'ai trouvé le cours de Mr. Brucker très clair, il explique les éléments essentiels à savoir et est accessible aux débutants.
Ce cours est disponible sous le chemin Cours/Web/Anatomie d'une URL, il est intitulé "Anatomie d'une URL".
Après avoir appris ce qu'était une URL, et comment elle était construite, il a fallu apprendre à les utiliser. Je pense que c'est plus un apprentissage passif que actif, je n'ai pas dédié de projets à l'apprentissage des URLs, pourtant elles apparaissent dans la quasi totalité de mes projets (que ce soit pour le clic d'un bouton, ou pour les commandes “fetch”).

Je trouve que ce format d'apprentissage est très bien, mais dédier une partie pratique aux URL peut être également intéressant, surtout si le cours n'est pas complet.

4.2 Serveur

Les serveurs sont essentiels au fonctionnement d'une page web. Ils hébergent les sites web, et traitent les requêtes des utilisateurs en fournissant des ressources.
Pour moi, l'interaction entre les sites web et le serveur à toujours été un concept complexe assez difficile à comprendre.
Dans un premier temps, j'ai appris ce qu'étaient les requêtes “http” grâce au cours de Mr.Brucker, et ce qu'était un serveur statique.
Ce sont les serveurs statiques qui m'ont permis de comprendre les subtilités de ce qu'était un serveur. En effet, dans un serveur statique, on créer nous même les interactions entre la page web et ses ressources, on peut regarder le serveur depuis l'intérieur.
Dans le cours de Mr.Brucker, il y a deux moyens de créer un serveur statique : en utilisant Node, ou en utilisant python. J'ai choisi d'utiliser python car il m'était plus familier que Node, de plus la mise en place du serveur est beaucoup plus simple sur python que sur Node car elle nécessite seulement d'entrer la commande “python -m http.server —-” avec —- le numéro du port que l'on souhaite utiliser, dans le terminal du dossier à partir duquel on veut créer le serveur statique.

Un serveur statique permet seulement d'héberger les sites statiques (avec seulement des fichiers statiques, tels que les fichiers HTML, CSS, javascript ou des images), donc c'était tout à fait adapté à mes besoins.
Pour comprendre ces serveurs, il est essentiel de passer par la pratique et de faire des projets.

4.3 Gestion des fichiers

Un élément central à la création et la gestion de pages web et dont on ne parle pas souvent, est la gestion des fichiers. En effet, il faut établir des règles communes afin de se simplifier la vie, et ce, dès le début du projet. Les règles de base sont détaillées dans le cours de Mr.Brucker. Par exemple pour commencer une page web, il faut créer un dossier au nom de la page, pour y ranger la page HTML (qui doit être nommée “index”) et d’autres fichiers associés à la page, essentiels à son fonctionnement (diverses ressources comme un fichier CSS ou javascript, ou des images). Une autre règle, c’est qu’il faut toujours utiliser des chemins relatifs. En effet, tout le site web est contenu dans un dossier, et ce dossier est destiné à bouger au sein de son ordinateur, ou être partagé sur un autre ordinateur. Si on utilise des chemins absolus, cela va soulever de multiples erreurs qui ne seraient survenues si on avait utilisé des chemins relatifs. Qu’elles soient conscientes ou pas, beaucoup de règles régissent notre façon de créer des pages web, elles permettent de généraliser des pratiques et nous simplifient la vie.

Un autre aspect de la gestion de fichiers est l’utilisation ou non de CDN (Content Delivery Networks). Un CDN est un réseau de serveurs répartis dans le monde, et qui permet de distribuer le contenu web rapidement. Pour utiliser Bootstrap par exemple, il faut récupérer le lien CDN pour le style et le script, ce qui donne :

<link href="https ://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/ bootstrap.min.css" rel="stylesheet" >

et

<script src="https ://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/ bootstrap.bundle.min.js"></script>

Il y a de nombreux avantages à procéder ainsi : on ne doit pas gérer les mises à jour, cela évite de télécharger Bootstrap sur sa machine, il n’y a pas de gestion de fichiers Bootstrap à faire, tout en restant rapide et efficace.

Mais on peut également procéder sans CDN. Il faut d’abord télécharger les fichiers sur le site getbootstrap/downloads. Puis dans la page HTML, en remplaçant :

<link href="https ://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/ bootstrap.min.css">

par

<link href="bootstrap/css/bootstrap.min.css" rel="stylesheet">

et

<script src="https ://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/ bootstrap.bundle.min.js"></script>

par

<script src= "bootstrap/js/bootstrap.min.js"></script>

On transforme alors le lien CDN en lien local. L'avantage majeur de cette méthode, est qu'on a un contrôle total sur les ressources qu'on utilise, ce qui est essentiel du point de vue de la sécurité. Par contre, cela a tendance à être un peu plus lent que l'utilisation de CDN.
Pour la gestion de fichiers, on a tout le dossier "Bootstrap" en plus à gérer.

Le dernier aspect de la gestion de fichiers que j'ai abordé pendant mon stage est NPM (Node Package Manager), c'est un outil incontournable pour les développeurs web de nos jours, il permet de gérer les dépendances d'un projet.
Pour cela, le cours de Mr.Brucker (Cours/Web/Projet front/Gestion des dépendances) m'a vraiment aidé, il est très clair et toutes les étapes sont bien expliquées. Je ne suis pas allé plus loin que ce cours pendant mon stage.
D'un point de vue gestion de fichiers, la méthode NPM rajoute quelques fichiers et dossiers (node-modules, package et package-lock), mais permet de mieux les gérer. On peut partager les dépendances de notre projet à n'importe qui, et NPM va automatiquement tout installer sur la machine de cette personne, elle permet aussi de mettre à jour les différentes versions des dépendances.

J'ai fait des projets pour utiliser les liens CDN, les liens locaux, et NPM, et cela m'a énormément aidé à comprendre.

5 Projet final

La dernière semaine de mon stage, Mr.Brucker m'a fait travailler avec Sarah Keghian, une autre stagiaire MPCI au LIS. Sarah avait travaillé sur la partie back-end et serveur. Ensemble, on avait toutes les compétences requises pour faire une page web.

Avec Sarah, nous avons choisi un jeu à créer parmis une liste que Mr.Brucker nous avait présenté. Notre choix s'est arrêté sur le jeu "Long Jump". C'est une adaptation du saut en longueur, le sport de décathlon, avec des dés. Les règles du jeu sont assez simples, vous pouvez les trouver dans la bibliographie [2] . Je vais tout de même rapidement résumer le jeu :

- Le jeu est constitué de deux manches : un "Run-up" où il faut faire le plus petit score possible (la limite maximale est de 8), et la partie "Jump", où il faut faire le plus gros score avec les dés qu'on a réussi à qualifier dans la première manche
- On a trois tentatives pour faire le meilleur score.

Sarah s'est occupée de rouler les dés et de faire une base de données. Pour ma part, j'ai codé l'interface, les règles du jeu et le tableau des scores.

5.1 GIT

La première étape du projet fut d'installer Git et Github. Github est une plateforme sur laquelle on peut collaborer sur des projets. Elle est essentielle pour travailler en équipe sur un projet de code commun car elle permet de créer des branches parallèles pour travailler sur un aspect du projet sans pour autant modifier la version des autres personnes de l'équipe. De plus, Github permet de garder tout un historique des différentes versions et actions réalisées. Cette plateforme est très utilisée dans le monde professionnel. C'est donc pour cela qu'il était intéressant d'installer github.

J'ai eu beaucoup de mal à comprendre toutes les subtilités de github, qui sont pourtant essentielles pour ne pas faire d'erreurs graves. Je me suis donc appuyé sur le cours de Mr.Brucker, qui m'a permis de comprendre le fonctionnement de chaque fonction de github. Nous avons également utilisé github desktop, qui, lorsqu'il est lié à une IDE, permet de "push" et "pull" les modifications faites (permet d'avoir la même version que les autres).

Au fur et à mesure que le projet avançait, j'étais de plus en plus à l'aise avec Github. Nous avons cependant rencontré quelques problèmes. Le plus gros problème était qu'on modifiait la base de données en même temps, il y avait donc deux versions de la base de données bien distinctes, et lorsqu'on essayait de les mettre en commun, Github Desktop envoyait des messages d'erreur. La solution à ce problème fut de mettre le fichier ".gitignore" à jour en y ajoutant la base de données et les logs.

Le fonctionnement de Github est difficile à comprendre pour tout le monde au début, donc je pense que c'est une bonne chose de l'apprendre pendant un stage, lorsqu'on a du temps à y dédier.

5.2 Fetch

Le front-end et le back-end sont entièrement dissociés, l'un travail sur la page web, alors que l'autre travail du côté du serveur. Pour le bon fonctionnement de la page web, il est essentiel que ces deux parties puissent communiquer. Comme dit précédemment, la "commande" fetch assure cette nécessité. Elle permet d'envoyer et recevoir des informations.

Avec Sarah, on a beaucoup utilisé la commande "fetch", que ce soit pour créer un groupe de dé, créer une partie, ou mettre à jour la base de données. Presque toutes les utilisations de "fetch" nous ont causé des problèmes. La première difficulté est d'envoyer ce que le serveur attend : un entier ou un fichier .json d'un entier par exemple. La deuxième difficulté est de savoir quoi faire de la réponse. Souvent, je recevais une réponse du serveur et j'essayais de l'utiliser, mais je ne respectais pas son format, et cela créait des erreurs. Une autre source d'erreur fut la gestion du temps de réponse du serveur. A quelques reprises, j'utilisais la réponse du serveur avant même de l'avoir eu.

Toutes ces erreurs sont simples à résoudre, mais sont dures à trouver. Javascript ne fournit pas le détail de l'erreur, et souvent il ne donne pas la ligne exact où l'erreur s'est produite Petit à petit, j'ai rencontré toutes les erreurs en relation aux commandes "fetch", et j'ai su les traiter. J'ai également croisé d'autres problèmes de code, mais c'était simplement des problèmes de logique. La commande "fetch" a été la principale source de problèmes.

5.3 Approfondissement de mes connaissances

Tout au long de mon stage, j'ai appris des éléments du front-end, d'abord comment faire une page grâce à HTML et CSS, puis comment le faire différemment, grâce à bootstrap. Finalement j'ai appris comment la rendre plus interactive grâce à Javascript. Mais je n'avais jamais lié toutes ces connaissances. Le projet final m'a fait me dépasser, j'ai dû faire beaucoup de recherches en parallèle pour résoudre mes problèmes.

Grâce à ce projet j'ai appris à "coder" en javascript. Avant je n'utilisais que des instructions basiques, alors que pour le projet, j'ai dû créer des variables, des boucles, des conditions... J'ai appris diverses méthodes, comme par exemple ouvrir une page HTML depuis une autre page HTML (le leaderboard), ou encore communiquer avec le serveur.

De plus, j'ai découvert l'univers du back-end. Comme je travaillais avec Sarah, je devais souvent interagir avec son code, et grâce à son aide, j'ai compris quelques éléments de Java, et de la construction d'un réseau.

6 Conclusion

Pour conclure, le but de ce stage a été de mettre en place un plan d'apprentissage efficace des langages web, en prenant en compte les difficultés rencontrées.

A travers ce stage, j'ai acquis une compréhension approfondie des langages web, notamment en HTML, CSS, et JavaScript. Je me suis également penché sur l'utilisation de bibliothèques comme Bootstrap, ce qui m'a permis de voir les outils que les développeurs web utilisent de nos jours. J'ai également compris le fonctionnement de gestionnaires de paquets tels que NPM. Ensuite, j'ai pu en apprendre plus sur la communication web et la gestion de fichiers grâce aux cours de Mr.Brucker. Finalement, j'ai découvert le back-end grâce au projet final, qui m'a également permis de voir toutes les étapes de la création d'un site web.

Le stage m'a encouragé à concrétiser les connaissances théoriques que je récoltais, en mettant en place des projets simples et compréhensibles. Les cours de Mr.Brucker m'ont accompagnés tout au long du stage en créant un fil conducteur, et la collaboration avec Sarah Keghian a enrichi l'expérience du stage en me faisant découvrir les liens étroits entre front-end et back-end.

Le stage a répondu à beaucoup de questions que je me posais, et m'a permis de bâtir des connaissances solides dans le front-end. Je trouve que l'expérience a été très bénéfique.

7 Remerciements

Je tiens tout d'abord à remercier Monsieur François Brucker qui m'a accompagné à chaque étape de mon stage malgré son emploi du temps chargé, et pour avoir été mon encadrant de stage.

Je remercie également le Laboratoire du LIS pour m'avoir accueilli dans son bâtiment.

Enfin, je remercie tous les chercheurs et stagiaires du LIS, qui m'ont partagé leurs connaissances et m'ont parlé de leur domaine d'étude.

Bibliographie

[1] MDN Web Docs. Mdn web docs : Resources for developers, by developers. https://developer.mozilla.org/.

[2] Dr. Reiner Knizia. Reiner knizia's decathlon. https://www.knizia.de/wp-content/uploads/reiner/freebies/Website-Decathlon.pdf.

[3] Dave Shea. Css zen garden : The beauty of css design. https://www.csszengarden.com/.

[4] W3Schools. W3schools online web tutorials. https://www.w3schools.com/.