POLYTECH DIJON

TRAVAUX DIRIGÉS

TypeScript

Promise / Fetch / Storage

OBJECTIFS

- Découvrir le langage TypeScript
- Utiliser les API `fetch` et `localStorage`
- Manipuler les promesses

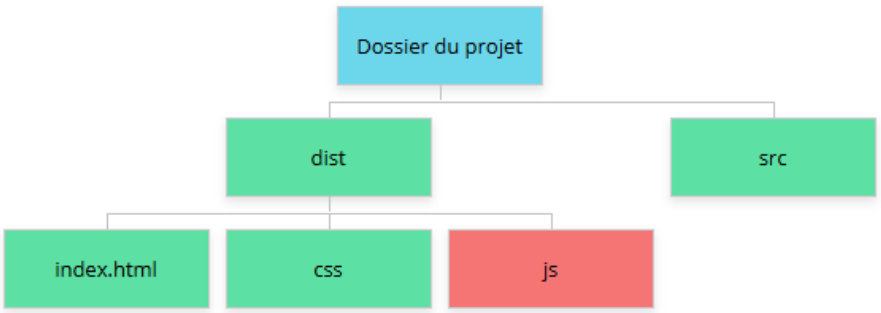
PREPARATION

NODEJS

- Si ce n'est pas déjà fait, installez NodeJS sur votre ordinateur.

PROJET

- Créez un dossier pour le projet du TD.
- Ouvrez ce dossier dans VS Code.
- Créez l'arborescence suivante au sein du dossier de votre projet :



> **Note**
> La couleur des dossiers dans l'arborescence vous sera expliquée un peu plus tard.

TYPESCRIPT

Pour ce projet, vous allez abandonner JavaScript dans un coin et travailler avec TypeScript. Pour rappel, TypeScript est un sur-ensemble à JavaScript qui apporte le typage statique, une meilleure intégration du paradigme objet, et tout un tas de petites choses fort sympathiques.

charles.meunier@ube.fr

Toutefois, les navigateurs ne comprenant que le JavaScript, il est nécessaire de passer par une phase de transpilation qui va convertir le code TypeScript en JavaScript compréhensible par Firefox, Chrome, Safari, ...

INSTALLATION

Installation locale au projet

- Ouvrez un terminal dans VS Code.
- Exécutez la commande suivante pour installer l'environnement TypeScript :

```
npm i -D typescript
```

NPM est le gestionnaire de paquets de NodeJS. Le principe est le même que celui d'APT sur Debian, à savoir centraliser les paquets disponibles pour l'environnement et gérer les dépendances de ces derniers.

L'option `i` indique que l'on souhaite installer un paquet

L'option `-D` indique qu'il s'agit d'un paquet utile uniquement pour le développement. Il ne sera pas nécessaire à l'exécution du programme lors de son déploiement en production.

Enfin, `typescript` est le nom du module que l'on souhaite installer.

- Vérifiez que l'installation de l'environnement TypeScript a bien fonctionné :

```
npx tsc -v
```

NPX est un outil permettant d'exécuter le code de paquets NodeJS. Ici, l'objectif est d'exécuter l'environnement TypeScript.

TSC est le nom du « compilateur » TypeScript.

L'option `-v` permet d'obtenir la version de TypeScript installée.

Installation globale

> **Note**
> Cette section est donnée à titre d'information.

Une installation locale au projet de TypeScript est intéressante si vous utiliser des versions différentes du langage dans vos projets. Ainsi, vous garantissez la stabilité de vos projets en limitant les impacts d'une nouvelle version de TypeScript sur du code plus ancien.

Par ailleurs, c'est également une solution utile si vous n'avez pas les droits pour installer TypeScript de manière globale sur l'ordinateur.

Toutefois, vous pourrez trouver plus intéressant d'installer TypeScript de manière globale sur votre ordinateur. Pour cela, utilisez la commande suivante :

```
npm i -g typescript
```

L'option `-g` réalisera une installation globale, c'est-à-dire en dehors du projet actuel. De cette façon, l'environnement TypeScript devient accessible depuis n'importe quel projet.

Après une installation globale, NPX n'est plus nécessaire pour exécuter l'environnement TypeScript :

```
tsc -v
```

CONFIGURATION

Une fois TypeScript installé, il est nécessaire de le configurer pour votre projet.

- Exécutez la commande suivante pour créer le fichier de configuration TypeScript par défaut :

```
npx tsc --init
```

> **Note**
> Si vous avez effectué une installation globale de TypeScript, il n'est pas nécessaire de préfixer la commande `tsc` par `npx`.

- Editez le fichier `tsconfig.json` qui vient d'être créé dans le dossier de votre projet. Ce fichier propose tout un ensemble de paramètres qui peuvent être appliqués à l'environnement TypeScript.
- Recherchez le paramètre `module` et affectez lui la valeur `ESNext`. Ceci sera nécessaire pour importer des modules plus tard dans le sujet.
- Recherchez le paramètre `rootDir` et configurez le avec la valeur `./src`. De cette façon, TypeScript cherchera les fichiers sources de votre projet dans le dossier `src`.
- Recherchez le paramètre `outDir` et configurez le avec la valeur `./dist/js`. Ceci permettra de ne pas mélanger les fichiers TypeScript avec les fichiers JavaScript qui seront générés à la transpilation. Ainsi, vous créerez vos fichiers TypeScript dans le dossier `src` et les fichiers JavaScript seront générés dans le dossier `dist/js`.

> **Note**
> Pensez à enregistrez le fichier `tsconfig.json` après modification et à décommenter les lignes que vous avez modifiées.

> **Explication**
> Le dossier `js` était présenté en rouge dans l'arborescence car son contenu est généré automatiquement. Prenez garde à ne pas y effectuer de modifications sans quoi elles risquent d'être détruites à la prochaine transpilation.

PREMIER PROGRAMME

ECRITURE ET TRANSPILATION

- Créez un fichier `main.ts` dans le dossier `src` de votre projet.
- Copiez le code suivant dans le fichier `main.ts` :

```
console.log("Hello World !");
```

- Transpilez votre projet avec la commande suivante :

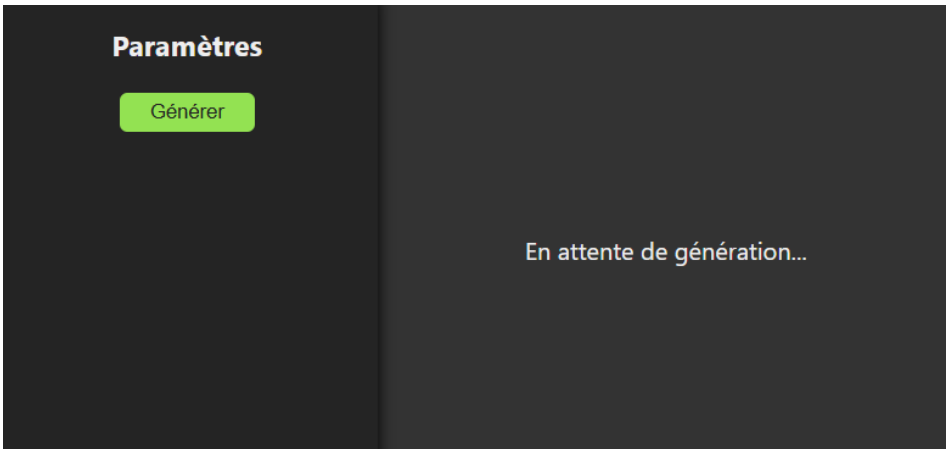
```
npx tsc
```

Rien de plus à préciser puisque vous avez indiqué à TypeScript où trouver vos sources avec le paramètre `rootDir`.

- Vérifiez que le dossier `dist/js` contient bien un fichier main.js
- Associez le fichier `main.js` à votre fichier `index.html`.
- Affichez la page `index.html` via Live Server et vérifiez le contenu de la console du navigateur.

HTML / CSS

- Modifiez votre fichier HTML et créez un fichier CSS pour obtenir quelque chose ressemblant à ceci :



- Modifiez le fichier `main.ts` pour que le message « Hello World ! » apparaisse dans la console lorsque l'utilisateur clique sur le bouton Générer.

> **Note**
> TypeScript est beaucoup plus exigeant que JavaScript quant à l'utilisation que vous faite de vos variables. Ainsi JavaScript vous laissera faire ce que vous voulez avec une variable qui peut contenir `null`. TypeScript non.

- N'oubliez pas de transpiler votre code avant de tester.

> **Astuce**
> Si vous trouvez pénible d'exécuter la commande `tsc` à chaque modification, vous pouvez ajouter l'option `-w`. De cette façon, `tsc` s'exécutera en mode « Watch », c'est-à-dire qu'il lancera automatiquement la transpilation des fichiers modifiés.

API

PRESENTATION

Nous allons utiliser une API prête à l'emploi dont vous allez afficher les résultats dans votre page Web.

L'API en question peut être interrogée directement depuis un navigateur via l'adresse suivante :

https://fractal.lesmoulinsdudev.com/express

- Ouvrez cette URL dans votre navigateur.

Vous reconnaissez ? Il s'agit de la fractale de Mandelbrot. Et durant cet exercice, vous allez créer un outil permettant d'explorer ce monde fascinant.

En effet, l'api `/express` accepte plusieurs paramètres pour définir le point exact de la fractale que l'on souhaite observer. Elle retourne en échange le code HTML de l'image générée (une balise `img` avec tout ce qu'il faut pour être intégrée à une page Web).

Voici la documentation de l'API :

URL	`/express`	
Méthode http	GET	
Paramètres	`xCenter`	*Coordonnée x du centre de la zone observée*
	`yCenter`	*Coordonnée y du centre de la zone observée*
	`zoom`	*Niveau de zoom*
	`maxIteration`	*Nombre d'itérations maximum*
	`colored`	*Coloration de la fractale (= 0 ou 1)*
	`red`	*Composante rouge (= 0 ou 1)*
	`green`	*Composante verte (= 0 ou 1)*
	`blue`	*Composante bleue (= 0 ou 1)*
Réponse du serveur		
Content-Type	text/html	
Body		

- Ajoutez `?zoom=200` à la fin de l'URL de votre navigateur et testez le résultat.

FETCH

Maintenant que vous avez vu le principe directement depuis le navigateur, vous allez procéder de même depuis l'intérieur de votre application grâce à la méthode `fetch`.

La méthode `fetch` permet d'effectuer des requêtes vers un serveur depuis un script JavaScript, ou TypeScript dans notre cas.

```
fetch("url_to_request").then((response) =>
{
    // Que faire s'il on obtient une réponse du serveur ?
}).catch((error) =>
{
    // Que faire si une erreur se produit durant la requête ?
});
```

`fetch` renvoie une promesse (`Promise`). C'est un objet JavaScript qui nous assure que nous aurons un résultat à la suite d'une opération asynchrone. Si l'opération se termine « normalement », la méthode fournie à `then` sera appelée. Si une erreur se produit durant l'opération, c'est la méthode passée en paramètre à `catch` qui sera exécutée. Mais dans tous les cas, nous aurons un retour.

Lorsque tout se passe bien, la fonction donnée à `then` sera appelée avec, en paramètre, un objet de type `Response` qui décrit la réponse du serveur à la requête. Attention, il ne s'agit pas seulement de la valeur renvoyée par le serveur, mais de toutes les caractéristiques de la réponse. Dont notamment le code HTML de retour.

```
if(response.ok)
{
    response.text().then((data) =>
    {
        // Que faire avec les données envoyées par le serveur ?
    }).catch((error) =>
    {
        // Que faire si une erreur se produit durant la récupération des
        // données ?
    });
}
```

- A partir des éléments précédents, effectuez une requête auprès de l'API présentée plus tôt et insérez l'image reçue dans la section principale de votre page web, lorsque l'utilisateur clique sur le bouton « Générer ».

  Prenez l'habitude de structurer vos programmes et éviter de mettre tout votre code dans une même fonction, ou pis encore, directement dans le script :

```
//Point d'entrée du programme
window.addEventListener("load", () =>
{

})

//Ajout d'un gestionnaire d'événement click au bouton
function initButtonEvent()
{

}

//Envoie une requête au serveur pour obtenir une image de fractale
function requestPicture()
{

}

//Ajoute le code HTML de l'image à la page
function insertPictureHTMLCode(htmlCode: string)
{

}
```

- Testez le bon fonctionnement de votre application.

ASYNC / AWAIT

Les promesses ont été un ajout majeur à JavaScript au temps des « callback of hell ». Mais leur enchainement ou leur imbrication a vite tendance à rendre le code difficilement lisible. C'est pourquoi, une syntaxe complémentaire a été introduite.

```
//Code avec then et catch
function requestAPI()
{
    fetch("url_to_request").then((response) => {
        if(response.ok)
        {
            response.text().then((data) => {
                doSomethingWithData(data);
            }).catch((error) => {
                console.error(error);
            });
        }
    }).catch((error) => {
        consle.error(error);
    });
}
```

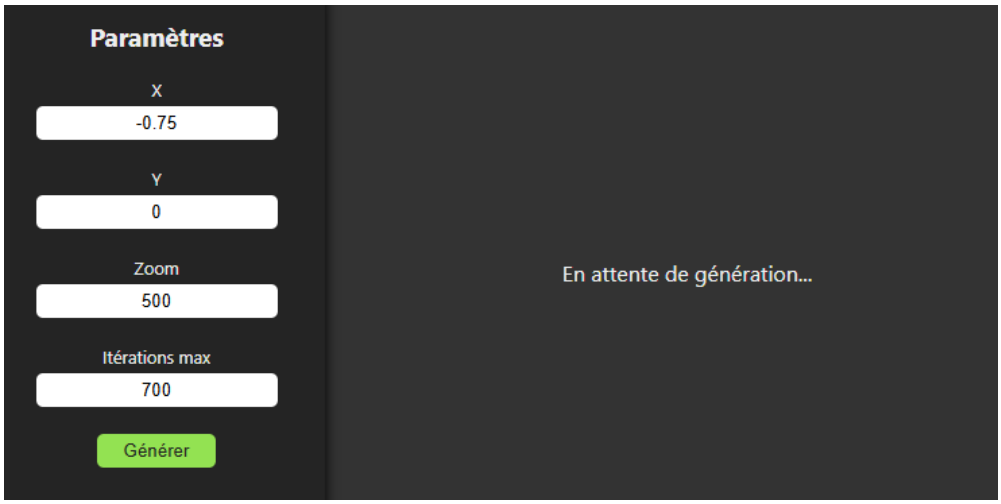
```
//Code avec async et await
async function requestAPI()
{
    try {
        const response = await fetch("url_to_request");

        if(response.ok)
        {
            doSomethingWithData(await response.text());
        }
    }
    catch(error)
    {
        console.log(error);
    }
}
```

Avec **async** et **await**, le code reprend une structure plus séquentielle tout en conservant un fonctionnement asynchrone.

- Modifiez votre code pour utiliser **await** et **async**.

PARAMETRES DE LA FRACTALE

- Modifiez votre page HTML pour qu'elle ressemble à ceci :

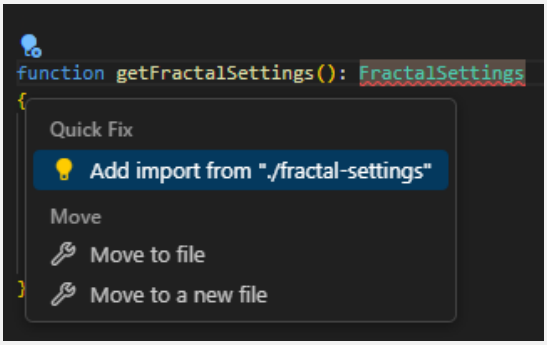


- Créez le fichier **fractal-settings.ts** dans le dossier **src** de votre projet.
- Dans ce fichier créez un nouveau type **FractalSettings** qui regroupera les paramètres (tous de type **string**) **xCenter**, **yCenter**, **zoom** et **maxIteration** de la fractale.

```
export type TypeName = {
    str: string,
    int : number
}
```

- Dans le fichier `main.ts`, ajoutez une méthode `getSettings` qui retournera un objet de type `FractalSettings` dont les attributs seront initialisés avec les valeurs des champs du formulaire.

> **Note**
> Pour pouvoir utiliser le type créé dans `fractal-settings.ts` vous devrez l'importer dans `main.ts`.
>
> **Conseil**
> Utilisez la petite lampe bleue de VS code pour importer vos dépendances.



> **Attention**
> Avec les mots clés `export` et `import`, nous utilisons les modules ES6. Ceci permet de n'avoir à lier notre page HTML qu'à un seul fichier (`main.js`). Tous les autres fichiers JS nécessaires seront ensuite intégrés automatiquement en fonction des imports réalisés. Toutefois, cela implique une petite modification dans la balise script de votre fichier `index.html` :
>
> ```
> <script src="script.js" type="module"></script>
> ```

- Modifiez le code de votre requête au serveur pour intégrer les paramètres de la fractale à générer (Query String).
- Testez le bon fonctionnement.
- Essayez les paramètres suivants :
  - xCenter : `-0.7491214101894764`
  - yCenter : `-0.125200239276915`
  - zoom : `100000000000000000`
  - maxIterations : `1000`

STORAGE

Il existe pleins d'endroits très sympathiques dans la fractale de Mandelbrot. Le problème est qu'il faut se souvenir des coordonnées.

Heureusement, votre navigateur met à votre disposition quelque mégaoctets d'espace disque pour conserver certaines informations coté client.

```
//Enregistrer une information dans le localStorage
localStorage.setItem("key_name", "value");

//Charger une information depuis le localStorage
const data = localStorage.getItem("key_name");
```

Notes

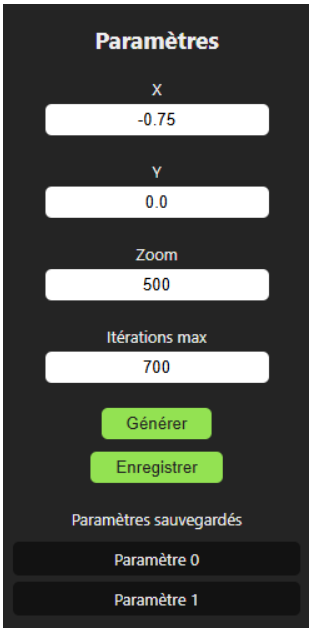
1. Le **`localStorage`** stocke des paires clé/valeur.
2. Le **`localStorage`** ne peut stocker que des chaines de caractères.
3. Si la clé demandée n'existe pas dans **`localStorage`**, **`getItem`** renvoie **`null`**.

Astuce

Le JSON permet de stocker une structure de données complexe sous la forme d'une chaine de caractères.

```
//Créer une chaine de caractères à partir d'un objet JavaScript
const jsonStr = JSON.stringify(jsObject);

//Récupérer l'objet JavaScript à partir de la chaine JSON
const jsObject = JSON.parse(jsonStr);
```

- A partir des éléments précédents, modifiez votre application pour que l'utilisateur puisse enregistrer les paramètres de la fractale et les retrouver dans une liste dès le démarrage de l'application :



- Testez le bon fonctionnement.

UN PEU DE COULEURS ?

- Modifiez votre application afin de permettre à l'utilisateur d'observer la fractale de Mandelbrot en couleurs :