Synnaxium Studio

TypeScript

Qu’est ce que c’est ?

Superset de Javascript

- Language Javascript…
- Mais étendu, avec des options supplémentaires

Système de typage

```ts
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}

const user: User = new UserAccount("Murphy", 1);
```

Try

Qu’est ce que c’est

TypeScript n’est pas reconnu par nos environnements natifs (navigateur, nodeJS, etc)

Nous allons devoir utiliser un “compilateur” pour le transformer en javascript

On peut déjà tester en utilisant le “Playground” de Typescript

https://www.typescriptlang.org/play

La vérification des erreurs ne se fait donc pas à l'exécution mais bien à la compilation

Pourquoi s’en servir ?

Limiter les erreurs lors de la programmation

- undefined, etc
- fautes d'inattentions : mauvais typages, etc
- meilleure documentation (via le typage)
- permet de convertir son code pour plusieurs cibles :
  - Le compilateur transformera votre code vers la version JS demandée

Installation

Tout d’abord nous allons installer et configurer NodeJS

Pour windows rendez-vous sur :

https://nodejs.org/en/

Pour utiliser typescript nous avons besoin de nodeJS pour la compilation et npm pour la gestion des packages et des projets

Installation

Une fois nodeJS installé,

Créez un dossier dans lequel vous allez mettre votre projet

Initialisez ce dossier avec la commande : npm init

Ensuite installez la dépendance “typescript” via la commande suivante :

npm install typescript --save-dev

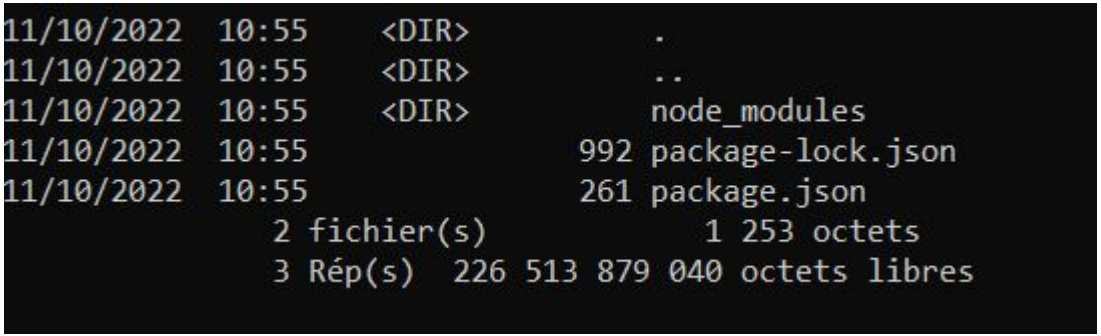
Jetons un oeil à notre structure de dossiers

Structure du dossier

On trouve 3 fichiers :

package.json => contient la définition de notre projet ainsi que la liste de ses dépendances

node_modules => dossier qui contient les librairies téléchargées



Créons notre premier script en Typescript

Pour cela :

Créez une page HTML qui comprendra une division, ainsi qu'un bouton.

L'objectif sera de lancer une fonction dès lors que l'on appuiera sur le bouton.

Nous allons passer par plusieurs étapes pour bien voir les aspects de base de typescript.

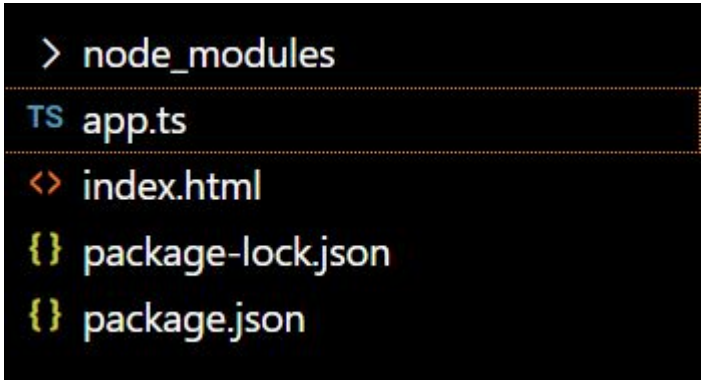
Une fois votre page créée, créez un script qui s'appellera app.ts

L'extension .ts correspond à un fichier "TypeScript"

app.ts

Commencez simplement par faire un “console.log(“Hello World !”);”

On devrait se retrouver avec une structure telle que celle-ci :



app.ts

Les fichiers .ts ne sont pas gérés nativement.

Pour cela nous devons créer l'exécutable, et donc lancer le compilateur.

Pour cela exécutez : *npx tsc app.ts*

npx ? npm ?

Node Package Execute

tsc : typescript compiler

Cela va convertir notre app.ts en app.js

app.ts

Ajoutez maintenant la partie script pour inclure votre fichier js

Le compilateur prend en charge beaucoup d'arguments possibles, ils sont disponibles à cette adresse :

https://www.typescriptlang.org/docs/handbook/compiler-options.html

Une bonne pratique est de créer un dossier src dans lequel nous ajoutons tous nos fichiers typescript, que nous compilerons dans le dossier "dist" ou "js"

app.ts

Pour compiler nous pourrons donc utiliser ensuite la commande :

npx tsc src/*.ts --outDir dist/

Pour éviter la répétition de cette commande à chaque fois nous pouvons configurer notre compilateur.

Pour cela, créez un fichier tsconfig.json

https://www.typescriptlang.org/tsconfig

Exemple

```
{
    "compilerOptions": {
        "outDir": "dist"
    },
    "files": [
        "src/app.ts"
    ]
}
```

app.ts

Si vous avez correctement écrit votre page ainsi que votre app.ts

Vous devriez pouvoir exécuter votre page html et découvrir le resultat tout devrait s'exécuter sans problème.

Si tout se passe bien, passons à la suite !

Les bases de Typescript

Maintenant que nous savons comment compiler nos scripts,

Regardons un peu ses spécificités

Je vous invite, à consulter la documentation si jamais vous avez des interrogations

https://www.typescriptlang.org/fr/docs/

Les bases de Typescript

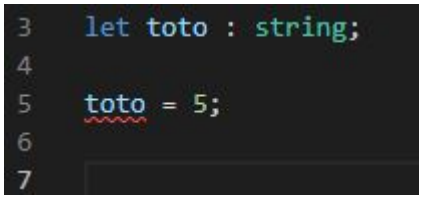
Javascript : Type inférentiels

Rend compliqué la lecture et possible les erreurs

Typescript : type explicites

Toutes les valeurs de retour et types de variables devront être précisés

Dès lors que le compilateur verra une erreur elle vous sera notifiée



Bases de typescript

Reprenons notre ancien script et notre page HTML.

Utilisez vos connaissances de base de javascript pour réaliser un script simple :

Créer une page HTML qui comprend une division et un bouton

Dans la division initialiser le contenu à 0

Dès que nous cliquons sur le bouton, incrémenter le compteur de 1

Définir les types

Pour définir un type il suffit de le mettre après un “:”

Voici les types de base de Javascript) :

boolean, bigint, null, number, string, symbol, et undefined

Typescript en ajoute quelques-un supplémentaires :

any : qui peut recevoir n’importe quoi
unknown : contrairement à any qui peut recevoir n’importe quoi, unknown va demander au type d’être spécifier avant tout utilisation
never : ne doit jamais arriver, code “impossible”

Les types

Voici un exemple qui illustre le *unknown*, nous allons le regarder ensemble :

https://www.typescriptlang.org/play#example/unknown-and-never

```ts
const jsonParser = (jsonString: string) => JSON.parse(jsonString);

const myAccount = jsonParser(`{ "name": "Dorothea" }`);

myAccount.name;
myAccount.email;

// If you hover on jsonParser, you can see that it has the
// return type of any, so then does myAccount. It's possible
// to fix this with generics - but it's also possible to fix
// this with unknown.

const jsonParserUnknown = (jsonString: string): unknown => JSON.parse(jsonString);

const myOtherAccount = jsonParserUnknown(`{ "name": "Samuel" }`);

myOtherAccount.name;

// The object myOtherAccount cannot be used until the type has
// been declared to TypeScript. This can be used to ensure
// that API consumers think about their typing up-front:

type User = { name: string };
const myUserAccount = jsonParserUnknown(`{ "name": "Samuel" }`) as User;
myUserAccount.name;
```

Les types

never :

Dans l’exemple précédent nous avons un exemple de *never*. Le principe est plus ou moins lié aux exceptions. Cela permet, via ce type de retour d'exécuter du code qui n’aurait pas dû être exécuté en temps normal.

Cela permet entre autres de fermer des ressources (fichiers, connexions, etc) avant d’arrêter l’exécution.

Les types

Création de types :

Dans typescript il est possible de créer des types, comme dans tout langage objet. Pour cela vous avez deux possibilités : les *interface* ou les *type*.

Il sera toujours préférable d'utiliser des *interface* par rapport aux "*type*" sauf si vous avez besoin de fonctionnalités spécifiques.

Le gros avantage des *interface* est qu'elles sont plus flexibles que les "*type*" (*type* s'appelle aussi alias)

Les Types : classes

JS supportant la création de classes et d'interfaces, les deux peuvent être combinés :

```
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}

const user: User = new UserAccount("Murphy", 1);
```

Try

Types structurels

Dans l'exemple précédent vous avez pu remarquer que, sans lien implicite nous avons pu faire rentrer un UserAccount dans un User

C'est parce que typescript et javascript font une égalitée "structurelle"

Les structures des données vont être analysées et si elles comportent les mêmes attributs elles sont alors égales

Types : composition / union

Vu la flexibilité de javascript, typescript permet de combiner des types.

On appelle ça la composition, par exemple pour un booléen il pourrait se traduire par :

```
type MyBool = true | false;
```

Le type MyBool est une composition de vrai ou faux

Cela fonctionne avec les interfaces et tous les autres types que nous pouvons créer.
https://www.typescriptlang.org/fr/docs/handbook/typescript-in-5-minutes.html#composition-de-types

Types : Casting / Assertion

Il est possible de caster ses variables pour forcer le type qui sera renvoyé dans une variable

Pour cela vous pouvez utiliser le mot clé “as”

Attention typescript analysera et vous renverra une erreur si ce type est impossible (par exemple ne fait pas parti de la composition)

L’autre possibilité pour faire une assertion est l’utilisation de <Type>

Strict et null

Typescript à deux modes de fonctionnement,

Le mot strict ou non strict

Le mode strict est celui que vous détestez car il tente de retirer un maximum de souplesse à javascript… mais aussi beaucoup d'ambiguité

Vous pouvez l'activer en ajoutant une ligne à votre configuration :

```
"compilerOptions": {
    "outDir": "dist",
    "strict": true,
    "target": "ES2017"
},
"files": [
    "src/app.ts"
]
```

Strict et null

A partir de maintenant, typescript viendra vérifier strictement toutes vos variable et vous empêchera de compiler s’il voit une potentielle erreur

Il viendra en l'occurrence faire des nullchecks, vérifier les initialisations, etc

Si vous souhaitez désactiver certaines fonctionnalités, vous pouvez consulter la doc de tsconfig.

Nullchecks

De notre côté, pour résoudre nos problèmes nous avons deux solutions

Soit faire un if(variable != null)

```
14  if(compteur != null)
15      compteur.addEventListener('click', test);
```

Soit utiliser l'opérateur ?

```
17  compteur?.addEventListener('click', test);
```

L'opérateur ? indique que la variable est optionnelle. Cet opérateur peut être utilisé dans des classes ou des fonctions

Exercices simples

Pour commencer et s’habituer aux différences je vous invite à réaliser les exercices suivants :

https://typescript-exercises.github.io